MASTER IN COMPUTER SCIENCE

Visualization software for 3D trabecular bones as a support for a diagnostic process a critical application of the Trident methodological framework

Octave, Michaël; Piedigrosso, Johan

Award date:
1997

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage, B-5000 Namur

Visualization software for 3D trabecular bones as a support for a diagnostic process : A critical application of the TRIDENT methodological framework

by

Michael Octave and Johan Piedigrosso

Thesis submitted in fulfilment of the requirements for the degree of
Master of Computer Science

Michael Octave
Johan Piedigrosso

Academic year 1996 - 1997

Visualization software for 3D trabecular bones as a support for a diagnostic process : A critical application of the TRIDENT methodological framework

Authors : Michael OCTAVE
Johan PIEDIGROSSO

Director : Dr Francois BODART, FUNDP, Belgium
Co-Director : Dr Tony S. KELLER, UVM, USA

Training period : Musculo-skeletal Lab
Mechanical Engineering College,
University of Vermont,
Colchester Avenue,
Burlington, VT 05405, USA.

Abstract :
The object of this thesis is to design and develop a Windows NT application for visualizing 3D objects, and particularly 3D representations of trabecular bones as a support for a diagnostic process. This application is aimed to be used by advanced users, in a research environment. It has been developed in the Mechanical Engineering Department of the University of Vermont, Burlington, VT, USA. This application is actually divided in two main programs, the first one is called 3D Surface Maker and is dedicated to create 3D representation of trabecular bones ; the second one is called 3D Viewer and is able to display several 3D objects, to rotate, transform and cut these objects.

Résumé :
Le but de ce mémoire est de développer une application Windows NT de visualisation d'os en 3D, et particulièrement leur représentation 3D comme aide au diagnostic. Cette application est dédiée à des utilisateurs expérimentés, dans un environnement de recherche. Il a été développé dans le Département d'Ingénierie Mécanique de l'Université du Vermont, à Burlington, VT, USA. Cette application comporte deux programmes distincts, 3D Surface Maker qui crée les fichiers sur les os et 3D Viewer qui permet l'affichage simultané de plusieurs objets 3D, la rotation, et autres transformations.

Keywords : 3D, imaging, C++, TRIDENT, HCI, interactive object.

Thanks

We want to give special thanks to people who helped us during our training
and the preparation and the redaction of our thesis. We spent about six months
in the Musculo-skeletal Lab of the Mechanical Engineering College,
in the University of Vermont, at Burlington, Vermont, USA.

Thanks to Dr Bodart and Dr Keller, our advisers,
who helped us to make clear the way we had to work,
Thanks to Dr Vanderdonckt, Mr Leheureux and Mr Leclercq for their invaluable advice,
Thanks to Dr Sullivan for the time he spent and Pascal Goossens for his advise,
Thanks to Mark, Rakesh, John, Michael L., Julie, Blanche, Anne and Michael G.
at the UVM and in Burlington. Their friendship and their fabulous welcome
made our training period a very rich professional and personal experience,
Thanks to Sophie for her support, her good advise and her patience,

Johan and Michael.

Table of Contents

Thanks **5**
Table of Contents 7
Table of Figures 9
Table of Tables 12
Table of Pseudo-Codes 12
Introduction **13**

PART I : The visualization process **15**

Chapter 1 : Visualization **17**
1. Definition and origins 18
2. Applications 19
3. Imaging, Computer Graphics and Visualization 19
Chapter 2 : Description of the process for the visualization program **21**
1. Introduction 22
2. Images 23
3. Display features 33
4. VTK Process - Surface construction 39
5. Volume construction 54
6. 3D Viewer - A visualization program 56
7. Conclusion 74
Chapter 3 : 3D graphics technical considerations **77**
1. OpenGL 77
2. The graphics hardware 78

PART II : A critical application of the TRIDENT methodological framework. .. **81**

Chapter 4 : First Dimension : Graphical User Interface Specifications **83**
1. Introduction 83
2. Task "analysis" 84
3. Expressing the product of the task analysis 104
4. Conclusion 134
Chapter 5 : Second dimension : Presentation Design From Ergonomic Rules **139**
1. Introduction 139
2. PU identification 140
3. Windows identification 141

4. AIOs selection 152
5. Transformation of the AIOs into CIOs 165
6. CIOs placement and manual edition of the presentation 166
7. Using an Expert System for Automatic Generation of User Interface 176
8. Conclusion 179
Chapter 6 : Third dimension : The software architecture derivation 183
1. Introduction 183
2. Architecture theoretical description 183
3. Hierarchies construction 187
4. Conclusion 207
Chapter 7 : Conclusion 211

Reference Books 215

Appendices 218

Appendix 1 : Object Model for VTK 219
Appendix 2 : Quantitative Computed Tomography 223
Appendix 3 : Magnetic Resonance Imaging 225
Appendix 4 : Main classes 227
Appendix 5 : VRML 241

Table of Figures

Figure 1-1 : Connections between Imaging, Computer graphics and Visualization 20
Figure 2-1 : Process Scheme 23
Figure 2-2 : Reading files in the whole process 24
Figure 2-3 : A 10x10 two-dimensional image array. 24
Figure 2-4 : Two two-dimensional arrays are seen as a three dimensional array 25
Figure 2-5 : Image Acquisition System 27
Figure 2-6 : A typical file format 28
Figure 2-7 : Displaying images in the whole process 34
Figure 2-8 : Conversion from 12-bit values to 8-bit values 35
Figure 2-9 : A 12-bit converted in 8-bit image and its corresponding histogram 35
Figure 2-10 : A slice of a trabecular bone 36
Figure 2-11 : Depth shading method applied to 50 slices 36
Figure 2-12 : 5 slices (left) and 10 slices (right) depth shading images 37
Figure 2-13 : Base image as displayed with simple depth shading 37
Figure 2-14 : Three other views of the object shown in Figure 2-13 38
Figure 2-15 : Cube sides 38
Figure 2-16 : The VTK surface construction in the whole process 40
Figure 2-17 : Cell type specification 40
Figure 2-18 : A drawing and its mesh representation (22 nodes) 41
Figure 2-19 : Approximation of a curved surface using polygonal facets [WATT96] 41
Figure 2-20 : Contouring a 2D image with a isovalue of 5 43
Figure 2-21 : Sixteen different marching squares cases 43
Figure 2-22 : Fifteen cases for marching cubes algorithm 44
Figure 2-23 : Construction of a cube with two slices 45
Figure 2-24 : A dialog box with a histogram (from 3D Surface Maker) 46
Figure 2-25 : Skin (a) and bones (b) selected regions 47
Figure 2-26 : CT slice through a human head and its corresponding position 48
Figure 2-27 : Contouring a CT scan and contouring bones only 48
Figure 2-28 : Marching cubes applied to two slices. (a) Top view. (b) Side view 49
Figure 2-29 : Mesh and rendered surfaces 49
Figure 2-30 : Dynamic Model applied to a basic visualization program 52
Figure 2-31 : Data flow diagram for the visualization process 53
Figure 2-32 : Volume mesh construction in the whole process 54
Figure 2-33 : A surface mesh (a) and a volume mesh from 3dmesh (b) 56
Figure 2-34 : 3D Viewer and its position in the whole visualization process 57
Figure 2-35 : A scene within its near and far planes 58
Figure 2-36 : Translation (a), Scaling (b) and Rotation (c) transformations 58
Figure 2-37 : Light spectrum 63
Figure 2-38: Ambient (a), diffuse (b) and specular (c) lights 64
Figure 2-39: Working with specular light 65
Figure 2-40 : Light positions 65
Figure 2-41 : The surface normal 66

Figure 2-42 : Defining vertices order as clockwise and counter-clockwise 67
Figure 2-43 : Two views for the same document [MSDEV96] 68
Figure 2-44 : 3D Viewer interface with multiple views 68
Figure 2-45 : Top, rear and right cutting planes 69
Figure 2-46 : Cutting planes 69
Figure 2-47 : 3D Viewer and an exported file in Netscape® 71
Figure 2-48 : Description of modules dependencies 74
Figure 2-49 : The whole visualization process 75
Figure 3-1 : Rasterization for 2 lines [WATT93] 79
Figure 4-1 : The toolbox metaphor. 84
Figure 4-2 : Microsoft Word seen as a toolbox 84
Figure 4-3 : The diagram of goal and sub-goal decomposition of the 3D visualization task 87
Figure 4-4 : The diagram of goal and sub-goal decomposition of the management of the scene sub-task 88
Figure 4-5 : The diagram of goal and sub-goal decomposition of the management of the objects sub-task 89
Figure 4-6 : The ERA model 106
Figure 4-7 : Graphical conventions for ACG 120
Figure 4-8 : Parallel functions 121
Figure 4-9 : ACG "Creation of a new scene" 122
Figure 4-10 : ACG "Selection of the current scene" 122
Figure 4-11 : ACG "Removing of the current scene" 122
Figure 4-12 : ACG "Changing the parameters of the scene" 123
Figure 4-13 : ACG "Geometrical transformation of all the objects in the scene" 124
Figure 4-14 : ACG "Cutting a part of the scene" 125
Figure 4-15 : ACG "Management of the lights" 126
Figure 4-16 : ACG "Saving into VRML format" 126
Figure 4-17 : ACG "Addition of an object into the current scene" 127
Figure 4-18 : ACG "Selection of the current object" 127
Figure 4-19 : ACG "Removal of the current object from the current scene" 127
Figure 4-20 : ACG "Changing the name of the current object" 128
Figure 4-21 : ACG "Getting information about an object" 128
Figure 4-22 : ACG "Changing the color of the current object" 129
Figure 4-23 : ACG "Changing the type of visualization of the current object" 129
Figure 4-24 : ACG "Showing the current object axis" 129
Figure 4-25 : ACG "Showing the current object box" 130
Figure 5-1 : Structure of the presentation 140
Figure 5-2 : Window W0 142
Figure 5-3 : Windows identification for the PU 1 142
Figure 5-4 : Windows identification for the PU 2 143
Figure 5-5 : Windows identification for the PU 3 143
Figure 5-6 : Windows identification for the PU 4 144
Figure 5-7 : Windows identification for the PU 5 145
Figure 5-8 : Windows identification for the PU 6 146
Figure 5-9 : Windows identification for the PU 7 147
Figure 5-10 : Windows identification for the PU 8 147
Figure 5-11 : Windows identification for the PU 9 148

Figure 5-12 : Windows identification for the PU 10 148
Figure 5-13 : Windows identification for the PU 11 149
Figure 5-14 : Windows identification for the PU 12 149
Figure 5-15 : Windows identification for the PU 13 150
Figure 5-16 : Windows identification for the PU 14 151
Figure 5-17 : Windows identification for the PU 15 151
Figure 5-18 : Windows identification for the PU 16 152
Figure 5-19 : Window W1-2 166
Figure 5-20 : Window W4-1 167
Figure 5-21 : Window W5-1 168
Figure 5-22 : Window W5-2 168
Figure 5-23 : Window W5-3 169
Figure 5-24 : Window W6-1a (standard window) 170
Figure 5-25 : Window W6-1b (Advanced options) 170
Figure 5-26 : Window W7-1 171
Figure 5-27 : Window W8-1 171
Figure 5-28 : Window W8-2 172
Figure 5-29 : Window W9-1 172
Figure 5-30 : Window W9-2 173
Figure 5-31 : Window W10-1 173
Figure 5-32 : Window W11-1 173
Figure 5-33 : Window W12-1 174
Figure 5-34 : Window W13-1 174
Figure 5-35 : Window W14-1 175
Figure 5-36 : Window W15-1 175
Figure 5-37 : Window W16-1 176
Figure 5-38 : Dialog box excerpt from 3D Viewer 177
Figure 5-39 : Dialog box proposed by SEGUIA 178
Figure 5-40 : Another way to place CIOs to add meaning 178
Figure 5-41 : A single line for each option 179
Figure 5-42 : Permanent window : solution 1 180
Figure 5-43 : Permanent window : solution 2 180
Figure 6-1 : Generic scheme of the architecture model 185
Figure 6-2 : Control objects hierarchy 186
Figure 6-3 : Hierarchy of functional objects for the tool "Creation of a new scene" 188
Figure 6-4 : Hierarchy of functional objects for the tool "Selection of the current scene" 188
Figure 6-5 : Hierarchy of functional objects for the tool "Removal of the current scene" 189
Figure 6-6 : Hierarchy of functional objects for the tool "Specifying the parameters of the current scene" 189
Figure 6-7 : Hierarchy of functional objects for the tool "Geometrical transformation of all the objects in the current scene" 190
Figure 6-8 : Hierarchy of functional objects for the tool "Cutting a part of the current scene" 190
Figure 6-9 : Hierarchy of functional objects for the tool "Management of the lights" 190
Figure 6-10 : Hierarchy of functional objects for the tool "Saving into VRML format" 191
Figure 6-11 : Hierarchy of functional objects for the tool "Addition of an object into the current scene" 191
Figure 6-12 : Hierarchy of functional objects for the tool "Selection of the current object" 192

Figure 6-13 : Hierarchy of functional objects for the tool "Removal of the current object from the current scene" 192
Figure 6-14 : Hierarchy of functional objects for the tool "Changing the name of the current object" 192
Figure 6-15 : Hierarchy of functional objects for the tool "Getting information about an object" 193
Figure 6-16 : Hierarchy of functional objects for the tool "Changing the color of the current object" 193
Figure 6-17 : Hierarchy of functional objects for the tool "Changing the type of visualization of the current object" 193
Figure 6-18 : Hierarchy of functional objects for the tool "Showing the current object axis" 194
Figure 6-19 : Hierarchy of functional objects for the tool "Showing the current object box" 194
Figure 6-20 : Primary hierarchy of control objects relative to the task : one occurrence of CO-W0 195
Figure 6-21 : Primary hierarchy of control objects relative to the task : several occurrences of CO-W0 196
Figure 6-22 : Pietri network for inter-PU dialog specification 209
Figure 7-1 : The toolbox model 211
Figure 7-2 : Development process for a toolbox model based application 212
Figure 7-3 : Common dialog box based application 213

Table of Tables

Table 4-1 : Parameters relative to the interactive task. 131
Table 4-2 : Parameters relative to the users stereotypes. 131
Table 4-3 : Parameters relative to the workplace. 133
Table 4-4 : Parameters relative to the users stereotypes 136
Table 5-1 : AIOs selection for alphanumeric data inputs 154
Table 5-2 : AIOs selection for boolean data inputs 154
Table 5-3 : AIOs selection for integer data inputs 154
Table 5-4 : AIOs selection for elementary data inputs 155
Table 5-5 : Transformation of the AIOs into CIOs 165

Table of Pseudo-Codes

Pseudo-code 2-1 : Display the first side of the selected region 38
Pseudo-code 2-2 : Display the left side of the selected region 38
Pseudo-code 2-3 : Marching squares algorithm 44

Introduction

We spent six months at the University of Vermont, in USA to develop a Windows NT application for 3D visualization of trabecular bones. This program was written in Visual C++ using several graphics libraries. At present, this program is used in the Musculo-skeletal lab of the Mechanical Engineering Department of the UVM.

This application is actually made of two different programs, the first one is called 3D Surface Maker and is dedicated to create 3D representations of objects, in this case pieces of trabecular bones; the second one is called 3D Viewer and is able to display several 3D objects, to rotate, transform, light and cut them.

This thesis is divided into two main parts :

First, we describe the visualization process, which includes description of images and how to display them, a summary of the Visualization Toolkit implemented by Schroeder, Martin and Lorensen, a 3D graphics library called OpenGL, a description of all 3D graphics principles needed in a basic 3D application and finally, a brief description of the 3D Viewer architecture.

The second part is aimed at carrying out an a posteriori *and critical application of the TRIDENT methodological framework developed at the University of Namur for the design of 3D Viewer, an interactive program. It is supposed to help to realize a non observed and weakly structured task (a diagnosis task). The program has the particularity of having permanent windows, case never handled with the methodology so far. Furthermore, a continuous comparison of this program with a toolbox is brung to the fore.*

Finally we have tried to adapt the TRIDENT approach for the development of applications, not necessarily 3D imaging ones, but with same characteristics as 3D Viewer : weakly structured task and permanent window.

PART I

The visualization process

The whole process of visualization is detailed in this part. We first develop the concept of visualization, its definition and origins, its applications and the distinction between Imaging, Computer graphics and Visualization.

Then we develop the process of the programs we wrote, in order to explain basics of imaging and visualization and few technical considerations.

Chapter 1 : Visualization

Visualization is an exploding domain at this time. It is being more and more used in so many different fields, and it is mainly thanks to advances in computer hardware and software technology. Computers are working faster and faster, graphic cards include special 3D chips for spatial transformations and memory is becoming cheaper and cheaper.

Even personal computers offer power to allow 3D graphics to everyone. New releases of Windows, such as Windows 95 and Windows NT support 3D graphics, with API[1] such as OpenGL[2] and allow easy programming and power of workstations.

[1] Application Program Interface: a set of libraries.

[2] Open Graphics Library. See below for more details.

1. Definition and origins

As [SCHROEDER96] says, "*we view visualization and visual computing as nothing less than a new form of communication. All of us have long known the power of images to convey information, ideas, and even feelings. Recent trends have brought us 2D images and graphics as evidenced by the variety of graphical user interfaces and business plotting software. But 3D images have been used sparingly, and often by specialists using specialized systems. Now it is changing*". The goal now is to extend current communication schemes, and to include 3D graphics in the communication, as it is for words, mathematical symbols and 2D images.

Visualization : "*The act or process of interpreting in visual terms or of putting into visual form*" [Webster's Ninth New Collegiate Dictionary]. The *expression* of complicated relations and equations is one magnificent step – *insight* gained from these relations is another. Today, computers with graphics can be used to produce representations of data from a number of perspectives and to characterize natural phenomena with increasing clarity and usefulness.
"Mathematicians couldn't solve it until they could see it !" [*Science Digest*, January, 1986, p. 49].

Visualization has different terminology. *Scientific visualization* is the field in computer science that includes the user interface, the representation of data, the processing algorithms and the visual representation. *Data representation* is more general than scientific representation, since it goes beyond the field of the scientists and engineers. Such data sources include financial, marketing or business data. It is now broad enough to include statistical methods and other standard data analysis techniques [ROSENBLOEM94]. A new trend of visualization is emerging now and is usually called *information visualization*, which ranges from the display of file/directory structure on a computer to the hyper-text documents on the World Wide Web.

The origin of visualization as a formal discipline dates to the 1987 NSF report *Visualization in Scientific Computing* [MCCORMICK87], but this field grown rapidly with many conferences, and the IEEE Visualization which is now well established, but the real origins are much older, around the eighteenth century with the arrival of statistical graphics even if it really exploded since the computer era.

2. Applications

The most famous application of the visualization is probably medical imaging, because it is probably the most impressive, because it can show the « inside » of the human body for example. Medical imaging is based on different techniques which will be described more precisely in a later chapter, but we can mention the *X-ray Computed tomography (CT)* and the *Magnetic Resonance Imaging (MRI)*. Both these techniques are ways for data acquisition and allow the capture of the internal anatomy of a living patient.

But the medical imaging is far away from being the only visualization application. The television and movies industry uses the computer visualization in more and more movies, such as Jurassic Park or the last Walt Disney's Toy Story movie.
The engineers also use the visualization in CAD applications like in the automobile domain or in the fluid simulation systems.[SCHROEDER96]

3. Imaging, Computer Graphics and Visualization

There is confusion surrounding the difference between imaging, computer graphics and visualization. According to [SCHROEDER96], we use the following definitions to set the differences :

- ***Imaging***, or image processing, is the study of 2D pictures, or images. This includes techniques to transform (e.g. rotate, scale, ..), extract information from, analyze and enhance images.
- ***Computer graphics*** is the process of creating images using a computer. This includes both 2D paint and draw techniques as well as more sophisticated 3D drawing (or rendering) techniques.
- ***Visualization*** is the process of exploring, transforming, and viewing data as images (or other sensory forms) to gain understanding and insight into the data.

As we can see, all these definitions are linked together, and we can summarize it like shown in Figure 1-1.

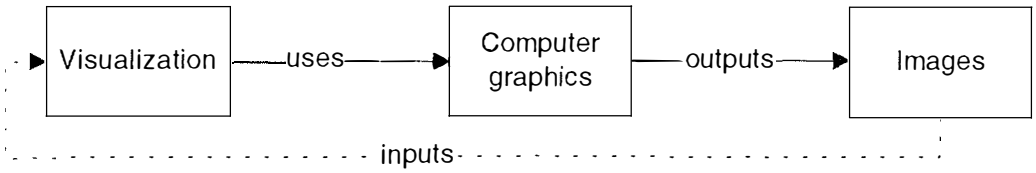


Figure 1-1 : Connections between Imaging, Computer graphics and Visualization

Images are usually the results of computer graphics, whereas the visualization *uses* computer graphics and their techniques to produce images. One can note that images can be the input data for the visualization process. The main differences between visualization and computer graphics are

1. **Visualization is most of the time in three dimensions**. It doesn't mean it doesn't work with data of two dimensions or lower, but it serves best for higher dimensions. We can for example easily imagine what a two-dimension array would look like with an image (computer graphics) but it is usually rather hard to understand a three-dimensional image representing the evolution of sales for the next year for ten different products.
2. **Visualization concerns itself with data transformation**. The meaning of the data is enhanced by the perpetual transformation of the information (rotations, zooming, ...)
3. **Visualization requires high interactivity** with the user, for all the processes of creating, transforming and viewing data.

In other words, one can say that visualization is an activity that encompasses the process of *exploring* and *understanding data.* [SCHROEDER96]

Chapter 2 : Description of the process for the visualization program

The process described below is the way the programs we developed are actually working. This chapter explains the whole process of visualization in the case of the Musculoskeletal System, however, it can be applied to any kind of data, as long as this one is in the right format.

This process includes data acquisition, data display, volume computation and visualization process in itself.

1. Introduction

The process starts from images. These images come from different kind of data acquisition such as Magnetic Resonance or Computer Tomography (§2.2 Images). Once these images are entered into computer the goal is to display them. Data is displayed on the computer screen and the user can select a region he is interested in (§2.3 Display features). The display features were used in our program in order to display MRI[3] as well as scanned images. The selection and three-view features were also implemented to give better idea of the shown image.

The selected region is finally saved as a VTK file[4] and the opportunity to display three different views of the same object is given to the user. The smaller region selected is then processed with filters and specials algorithms for 3D surface construction. A mesh file, that is, a net of points and lines is then created and can be used in any 3D program such as CAD[5] (§2.4. VTK Process - Surface construction). Using the library, we developed a straightforward small program to avoid script editions[6].

The next step is to fill the surface created with VTK to build a volume, which is carried out with a UNIX-based program (§2.5 Volume construction). This program was written by John Sullivan [SULLIVAN95] and is command-line oriented. The only problem we found out was the incompatibility between all different files we handled. A volume mesh file is created and can be compared with the initial 3D surface mesh.

---

[3] Magnetic Resonance Imaging. See below for details.

[4] Visualization Toolkit. This is the library developed by [SCHROEDER96] which we used for this part. It includes a lot of algorithms and will be detailed later in this chapter.

[5] Computer Aided Design

[6] The 3D Surface Maker program we developed is based on VTK library but is written in the Visual C++ environment which is much more powerful than the script language provided with VTK.

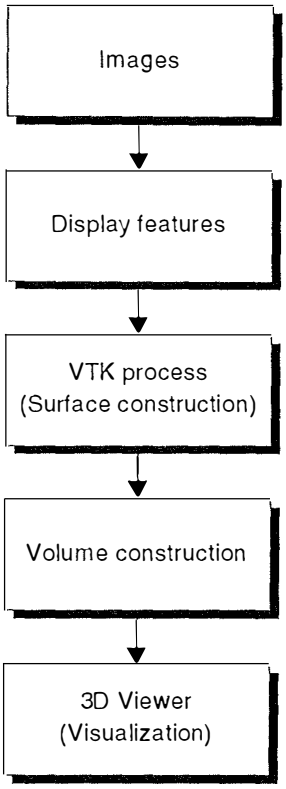


Figure 2-1 : Process Scheme

A decision is then necessary to accept or reject the volume mesh, depending on the quality of the 3D volume mesh. Indeed, 3D Viewer allows several surfaces and volumes to be displayed in the same time and in the same coordinate system, so they can be compared (§2.6 3D Viewer, a visualization program). If the volume seems to fit to the initial surface, the volume mesh is kept and can be used for numerical analysis. 3D Viewer was fully developed in Visual C++ for 5 months at the University of Vermont. A whole section will be dedicated to graphics to describe first basics of the components of 3D graphics such as lights or colors then we will describe briefly a graphic library on which we based to develop 3D Viewer and we will close this section with a description of the architecture of the software. Each step shown in Figure 2-1 corresponds to a section in this chapter.

2. Images

This section will develop basics of images and their source. Then we will briefly talk about file formats since it was and remains a problem between programs and particularly the one we

wrote. Finally we will discuss the problem of trabecular bones since it is the problem we looked into. We will show different images of trabecular bones and show how visualization can help a diagnostic. As shown in Figure 2-2, the goal of this section in the whole chapter is to read files, and particularly binary files.

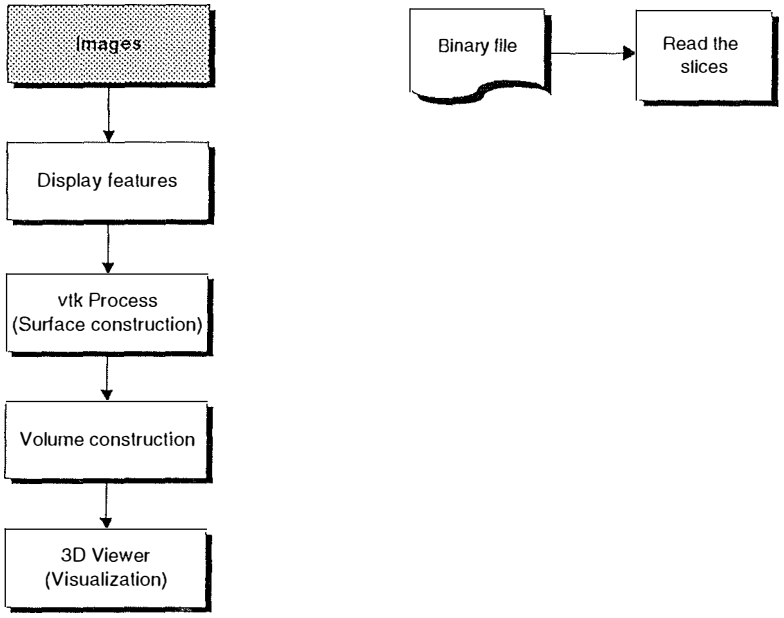


Figure 2-2 : Reading files in the whole process

2.1 Two-dimensional image arrays - Raster data

A two-dimensional image array is simply a matrix of n columns of m rows, where each element is a point that can be of any color. Figure 2-3 shows an example of a two-dimensional image array [SCHROEDER96].

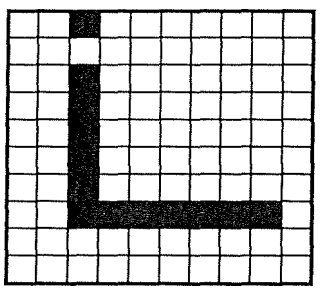


Figure 2-3 : A 10x10 two-dimensional image array.

A raster format breaks an image into a grid of equally-sized pieces, called pixels, and records color information for each pixel [JASC96]. In Figure 2-3, every square is a pixel, some are

black and the others are white. In that case, the image is only black and white, so each pixel is 1-bit encoded (0 is black and 1 is white). The more colors used, the more bit needed to encode each pixel. The problem of files encoding will be discussed later.

2.1.1 Voxel Data Sources

When you are talking about two-dimensional images, each element is called pixel. In 3D world, any element that is part of a 3D object is called a *voxel* [WATT93], [SCHROEDER96]. Voxel stands for *volumetric pixel.* These voxels are usually built as a sequence of two-dimensional images, and therefore are seen as three-dimensional arrays, as shown in Figure 2-4.

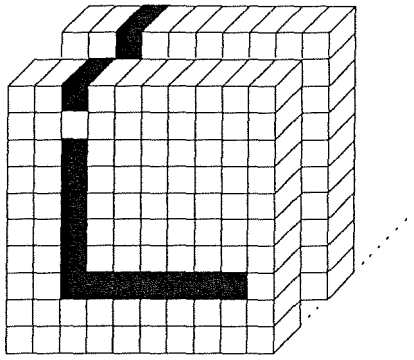


Figure 2-4 : Two two-dimensional arrays are seen as a three dimensional array

In medical imaging, data are usually acquired as two-dimensional images, but more and more software are at present developed to build a 3D view of multiple 2D images. One of the algorithms used to achieve a 3D view — The Marching Cubes Algorithm [LORENSEN87] — will be discussed later. This section will explain the main three ways of data acquisition in medical domain. They are Quantitative Computed Tomography, Magnetic Resonance Imaging and Quantitative Serial Sectioning. It is based on [BRIGGS95].

A. Quantitative Computed Tomography

Computer Tomography (CT) was first developed for intracranial imagery in the late 1960's by Godfrey Hounsfield. Since then major technical advances have resulted in substantial improvements in image quality and a marked reduction in scanning time. Tomography has been used extensively in diagnostic radiology prior to the introduction of CT. The structures in the tomographic plane remain in focus while those in planes above and below are blurred out. CT and tomography differ in that CT has the ability to detect more subtle differences in the absorption and attenuation of x-rays than is possible with tomography [BRIGGS95].
For more details on CT, see Appendix 2. [BRIGGS95]

B. Magnetic Resonance Imaging

Medical Resonance Imaging (MRI) has rapidly become widely discussed regarding its influence on medical imaging. For more details on how Magnetic Resonance (MR) signals are generated and detected, how an image is formed, what general sorts of tissue properties can influence the signals and thereby give rise to tissue contrast and how the machine parameters can be used to manipulate the tissue contrast observed in the image, see Appendix 3. [BRIGGS95]

C. Quantitative Serial Sectioning

(QSS) This method is using a camera to digitize sections at a reasonable resolution, which usually range from 256X256 pixels at a resolution of 0.2mm (i.e. 5 pixels every millimeter). Trabecular bones images used in our programs are generally coming from this kind of data acquisition. Since this kind of data acquisition is mainly used in the Musculo-skeletal lab of Tony Keller, we will develop more precisely the way it works.

1. Specimen preparation

Human lumbar spines are harvested during routine autopsies. Some 9mm x 9mm x 9mm cubic cancellous bone specimens are prepared from the vertebral centrum using a low-speed diamond saw. Selection of the samples is regionally random. However, the vertebral regions that have apparent defects in the continuity of trabeculae due to blood vessels and bone diseases are avoided. Bone specimens are irrigated with 0.9% saline during machining and following technical testing, and are stored frozen at -30°C.

The bone specimens are thawed at room temperature for two hours before mechanical testing. Using a MTS 858 Bionix™ test system, each specimen is nondestructively loaded in compression ($\in_{max}$ = 1%) along three orthogonal axes corresponding to the superior-inferior (SI), anterior-posterior (AP), and medial-lateral (ML) axis. The surfaces of stainless-steel load platens are polished to a surface flatness of 2 $\mu m\ cm^{-1}$ and lubricated prior to testing each specimen. Load and displacement are recorded at 1 kHz using a Nicolet™ 430 digital oscilloscope. Displacement are measured by means of crosshead movement and are corrected for the test machine compliance. A stress-strain analysis program was developed to correct the recorded displacement, and to determine the elastic modulus, *E*. Elastic modulus is computed from the slope of the stress-strain curves using a strain range of 0.1 - 0.8%.

After mechanical testing, the bone marrow is removed from the specimens using a high pressure water jet and defatted with several acetone washes and rinses. The marrow-free samples are then dried in a furnace at 100°C for 1 hour, and weighted on a Mettler AE 163 (Hightstown, NJ) analytical balance. Apparent dry density of the specimens is calculated as the ratio of the dry weight to the cube volume, the latter measured using a caliper (± 0.025mm).

Specimens are then bleached using 3% hydrogen peroxide, embedded in black-colored polyester resin and centrifuged at 1000 rpm. The centrifuge process facilitates infiltration of the polyester resin into the pores of the cancellous bone samples. [GOOSSENS95]

2. Imaging process

In order to obtain a detailed understanding of the bulk variations in bone structure, bone specimens are serially sliced along the superior-inferior axis every 20 µm/pixel using a Reichert-Jung® polycut E microtome. At an image resolution of 20 µm/pixel, 16-bit color video images of each sectioned surface are recorded using an image acquisition and analysis system. This image system shown in Figure 2-5 consists of a CCD camera and a Pentium P120 computer with a TARGA™ 16 graphic board and a MIPS program. The TARGA board is able to convert the image of the CCD camera into a digital screen display of 510 x 480 pixels with 32,768-color resolution. A total of 250 planar digital images spanning 5 mm in depth for each specimen are obtained producing a 3D 16-bit image array comprised of approximately 50 million 20-µm voxels. For the remaining cancellous bone specimens, only the orthogonal surfaces are imaged. The 16-bit images are thresholded into white (bone) and black (marrow) binary images. [GOOSSENS95]

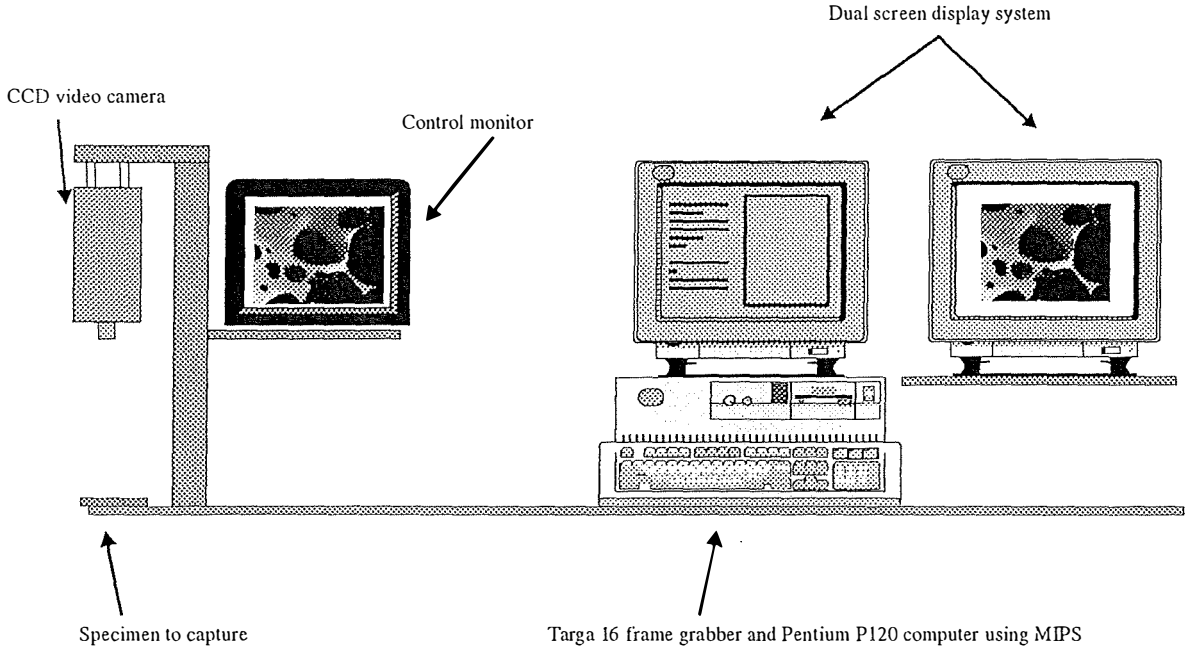


Figure 2-5 : Image Acquisition System

2.1.2 File format and sizes

One of the main problems with imaging files is their size. Assuming for example a file of 512X512 pixels, where every pixel is encoded with 12 bits. The size of this file, without any header, will be 524,288 bytes – because even though each pixel is 12-bit encoded, it must be saved in 2-byte words. Of course, recent developments in data compression have allowed to drastically reduce the size of files, however, it remains a serious concern. So far, many medical images are still not compressed, to make data exchange easier, even if they must have a particular format.

The format of a file is very important, because it allows interchange of files between people and programs. Each file should include a header and a data section. The header usually includes a file format version, a description of data (number of bit per pixel, x and y dimensions, ..) and the structure of data (e.g. the data may be compressed or not). The data is then added, and can be in several groups. For example, with a 3D object, all the points describing an object are defined first, followed by the connections between these points. A model of a file format is shown in Figure 2-6.

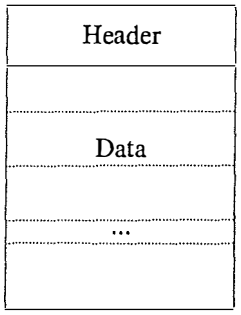


Figure 2-6 : A typical file format

Files coming from MRI or CT scans are usually made up of many *slices*, representing two-dimensional images. These files are structured as shown in Figure 2-6. The header includes the number of bits per pixel and the width and length of the image. Every pixel is then stored in 2 bytes, because it is encoded in 12 bits, describing a gray-scale value ranging from 0 to 4095, one after the other. The number of bit per pixel is particularly important when it has to be displayed. For example, let us take an image of 12 bits par pixel, one pixel being stored in 2 bytes. All the values in data file are ranging from 0 to 4095. But one could say every pixel is using 16 bits, but ranging also from 0 to 4095. In the first case, every single value ranging from 0 to 4095 represents one and only one gray-scale value, but in the second case, values from 0 to 15 represent the same gray-scale value because 16-bit value stores 16 times more information then 12-bit value.

Files we used coming from QSS are 1-bit value pixels. It means they are storing black and white images. Moreover, the whole set of slices is stored in the same file, so they can be huge. The format of these files is different, because they are storing a particular set of data, trabecular bones. Instead of storing every pixel of the image, it is only storing the coordinates of white pixels (value = 0). So if one takes a look at the file, he will find sets of three values : (`slice_number`, `x_coordinate`, `y_coordinate`). Even if that format does not need any specific delimiter between each slice (because of the `slice_number` parameter), the size of the file is much bigger, because of redundant information. Indeed, for every 2 values, a third one is stored, which takes a third more room.

2.2 Trabecular bones

This section is based on an article written by T. Keller and T. Hansson. Some paragraph have been shortened but the interested reader can consult [KELLER95] for more details.

2.2.1 Introduction

Osteoporosis, which is characterized by a reduction in skeletal bone mass and concomitant change in skeletal structure, produces an increased risk of fracture in patients and thus has a devastating effect in terms of morbidity, mortality and cost of health in our increasing senile population. Osteoporosis affects both the appendicular and axial skeleton of adults, and is a well recognized public health problem of increasing proportions. Over 1.2 million fractures occur in the United States each year, including over 500,000 cases of vertebral fracture, and 200,000 cases of hip fractures, one third to one-half of which occur in women over the age of 65. In the United States, the personal and medical costs associated with osteoporotic fractures are expected to increase dramatically in the next two decades, since the number of individuals over the age of 65 has been predicted to double by the year 2010 (1983 United States census).

A close association between bone mineral loss due to osteoporosis and the risk of fracture has been clearly established. Skeletal structures, such as the vertebral bodies and proximal femur, which are compromised primarily of trabecular bone appear to be particularly at risk. Thus, development of clinical diagnostic tools sensitive enough to identify imminent fracture or collapse of vertebral bodies and other weight-bearing tissues is essential. Until these tools are developed, the ability of a clinician to clearly evaluate a patient's bone status, prevent osteoporosis or determine the effect of therapeutic treatments is severely limited.

2.2.2 Epidemiology

The aging skeleton is characterized by a gradual loss mass which decreases bone strength (force or stress at failure) and increases fracture risk. A more rapid loss of bone mass occurs in post-menopausal women, and collectively these processes are referred to as primary osteoporosis. At present time the precise etiology osteoporosis is unknown. Because of increase morbidity and immobility produced by hip fractures, many epidemiological studies of osteoporosis have focused on hip fractures. Until recently, vertebral fractures were deemed to be of lower incidence and concern than hip fractures. There are, however, no reason why increases in the incidence of hip fractures should not reflect a similar increasing incidence of osteoporotic spine fractures. A recent Swedish study found that 43 % of the subjects who had a hip fracture also had one or more vertebral fractures of an osteoporotic type [ZETTERBERG90]. Of note, is the fact that the prevalence of osteoporosis seems to have become more and more common, particularly in industrialized countries. This increase is partly explained by the fact that populations in most industrialized countries are growing older but also by an increased risk. The osteoporotic vertebral fracture is probably the most frequent of all fragility fractures, particularly if every vertebral fracture in the spine is considered.

Vertebral fracture is about four times more common in women than in men and the risk for a vertebral fracture has been found to increase almost exponentially with age. The frequency of osteoporotic vertebral fracture also increases during menopause in women. From this point on there is a steady increase in vertebral fracture frequency throughout life. In this respect, the vertebral fragility fractures differs from fractures of the distal radius. The prevalence of the latter increases at the same age as the vertebral fracture, but levels out after 60-65. An interesting recent finding is that the increase in risk for a fragility fracture between 1985 and 1991 was almost twice as high for men as for women [ZETTERBERG94]. Depending on the age groups studied (40 to 80+ years), the prevalence of osteoporotic vertebral fractures varies from 5% to somewhat over 50%.

Radiographically detectable compression fractures of the spine for most clinicians has verified the presence of osteoporosis or bones fragility. Without any known pathomorphological aberrations distinguishing osteoporotic bone from non-osteoporotic bone tissue, the fracture itself defines pathology. Since the occurrence of a fracture is not only the result of the mechanical properties of the bone, but is also a function of the fracturing trauma, both factors must be considered when defining osteoporosis. In the presence of a patient with a recent fracture, knowing nothing or very little about the patients bone quality or the forces involved in the trauma, the most practical way for clarifying whether a fracture is osteoporotic or not, is Harold Frost's criteria of the "everyday trauma". Frost stated that a fracture occurring as a

consequence of an everyday trauma indicates that the patient has osteoporosis or bone fragility. Even if the technology today allows us to determine, for example, the amount of bone mineral in different parts of the human skeleton we still lack practical techniques for measuring the fracture generating forces. Therefore the "everyday trauma" definition is still a practical measure for estimating bone fragility [FROST93].

Without any distinct differences between the bone tissue in the osteoporotic versus the normal subject there are, however, apparent difficulties in assessing the limits for normality. Since demineralization of the human skeleton is usually a more or less continuous process from relatively early in life, weakening of the skeleton is a part of normal life and aging. An osteoporotic or fragility fracture occurs in those subjects in which the demineralization progresses to a level where the spine or other parts of the skeleton no longer can resist an everyday trauma. In many subjects with spinal osteoporosis the vertebrae may become so demineralized that they can not resist the spinal loads accompanying everyday life. Since the amount of bone mineral in combination with the loading conditions determine the occurrence of a fracture, a subject with a *low amount of bone mineral*, but *no fracture*, has *osteopenia*. A subject with a *low amount of bone mineral* and *a fracture* sustained during a minor "everyday" trauma is likely to have *osteoporosis*.

2.2.3 Basic bone physiology

Bone is a two-phase, porous, directional composite material, comprised of hydroxypatite (inorganic or mineral phase) and collagen (organic phase). In the normal adult skeleton, hydroxyapatitie constitutes approximately 2/3 of the weight or about 50% of the volume of dry bone tissue. Bone composition can be described by several histologic variables, including mineral content, porosity and density. The density (mass/volume) may refer to either the wet or dry bulk density (mass per unit volume of a region of bulk bone). Bone devoid of pores has a tissue density or specific gravity of approximately 2g/cm^3. Bulk density or apparent density (ρ_a), however is a mesure of both the porosity and mineral content of bone and range from < 0.1 g/cm^3 to approximately 2.0 g/cm^3.

All of these histologic variables have been used to describe the composition of bone. From a morphological point-of-view, two principal types of bone are recognized: **cortical** and **cancellous**. In the adult skeleton, both cortical and cancellous bone have roughly the same amount of mineral except in metabolic diseases such as *osteogenesis imperfecta* for which the mineral content is significantly reduced. Cortical or "compact" bone is generally distinguished from cancellous or **trabecular** bone by its lower porosity (<30% pores by volume) and higher apparent density (> 1.7 g/cm^3), and is most prevalent in the shafts of long bones. The ends of

long bones and the axial skeleton (spine) are comprised primarily of trabecular bones, which in the case of the axial skeleton has a porosity greater that 70 % or an apparent density less than 0.6 g/cm^3. By virtue of its inherent porosity, trabecular bone has an extremely complex structure or "architecture". Decreases in bone mass associated with aging, inactivity and menopause have profound effects on the architecture of trabecular bone. Collectively, the changes or "adaptations" in skeletal mass are architecture are referred to as modeling and remodeling processes.

2.2.4 Vertebral strength

The spine is a weight bearing structure which, besides protecting the spinal cord and offering exceptional flexibility and range of motion, must continually support the weight of the torso and head. Together with everyday activities, these structures must support to a significant degree, axial compressive forces on the vertebrae and intervening disc tissues. In the L1-L4 lumbar spine this amounts to approximately 50-60% of the subjects body weight. Consequently, numerous investigators have examined the axial, compressive strength properties of cadaveric human thoracolumbar vertebrae. Ultimate strength values ranging from about 1 to 15 kN have been recorded in these experimental studies, most of which have examined tissues from more age subjects (eg. > 40 years). To which extent the inability to obtain specimens representative of the entire population has influenced these strength values is hard to estimate. However, it is reasonable to assume that the compressive strength of vertebrae is grossly underestimated for ages below 50 years. Experimentally, as well as clinically, large variations in bone strength have made it very difficult to define a specific threshold or even a range with which to differenciate normal bone from osteoporotic bone. The latter also requires knowledge of the physiologic forces and stresses which act on the vertebral structures.

2.2.5 Vertebral morphology

Although non invasive measures of bone density are now considered the most effective method known for predicting fracture risk, these techniques appear to be only about 70% accurate. Presumably, other material features of bone and supporting structures are needed to explain the additional 30% of causes of fracture risk. The additional factors play a greater role in the spine, making bone density less predictable in the spine than in other regions of the skeleton which are comprised of more dense bone. Trabecular bone researchers currently attribute the unexplainable variation in mechanical properties to differences in the morphological features of this tissues.

In vertebrae, large variations in trabecular density and mechanical properties have been noted within adjacent regions separated by only a few millimeters [KELLER89], [KELLER92]. Five morphological distinct regions of trabecular bone are found in the vertebral centrum : a superior, 1st transitional, center, 2nd transitional and inferior level. The superior and inferior sections each occupy approximately 30-35% of the total segment height and exhibit patterns of orientation distinct from the center and transitional sections. The transitional and middle portions of the centrum consisted primarily of the plate-like trabeculae forming a closed cell structure in contrast to the superior and inferior sections of the centrum which consisted primarily of rod-like trabeculae forming an open cell structure. Trabecular bone structure is more dense in the inferior and superior sections than in the central sections of the lumbar centrum. Plate-like trabeculae are associated with the central regions of less dense bone. The central, plate-like regions of the lumbar spine, therefore, appear to be somewhat unique in terms of it trabecular architecture. The functional significance of this finding remains to be determined.

The complex organization and distribution of vertebral trabeculae and trabeculae in other regions of the skeletal support the generally accepted hypothesis that function directly influences the structure and strength of bone, a relationship known as Wolff's Law [WOLFF1892]. From a mechanical engineering standpoint, trabecular bone behaves similarly to porous engineering materials due to its cellular structure and large energy absorption capabilities. The distribution of trabecular bone density and mechanical properties within vertebrae varies along the axis and within the cross section of vertebrae. Some investigators have reported a variable or heterogenic distribution of trabecular bon tissue physical and mechanical properties for the vertebral centrum [KELLER89], [KELLER92], [KELLER93]. Most of these studies have noted that anterior regions of the vertebral centrum are generally less dense and less strong than posterior regions. Keller and associates [KELLER92] noted that the superior and inferior regions of lumbar vertebrae are denser than the central and transitional regions.

3. Display features

Once data has been digitized and read into the computer, it has to be displayed on the screen. This section will explain how a binary file, and a set of two-dimensional images can be displayed on a monitor. The first part will describe the way to display a gray-scale image, using the useful bits from an image, the second one, an easy method called *Depth shading*. The third part will explain the selection process of an area in the image and the usefulness of the visualization in the selection process. Technical considerations will be discussed in Chapter 3.

Figure 2-7 shows the goal of the current section and its contribution in the whole process of visualization.

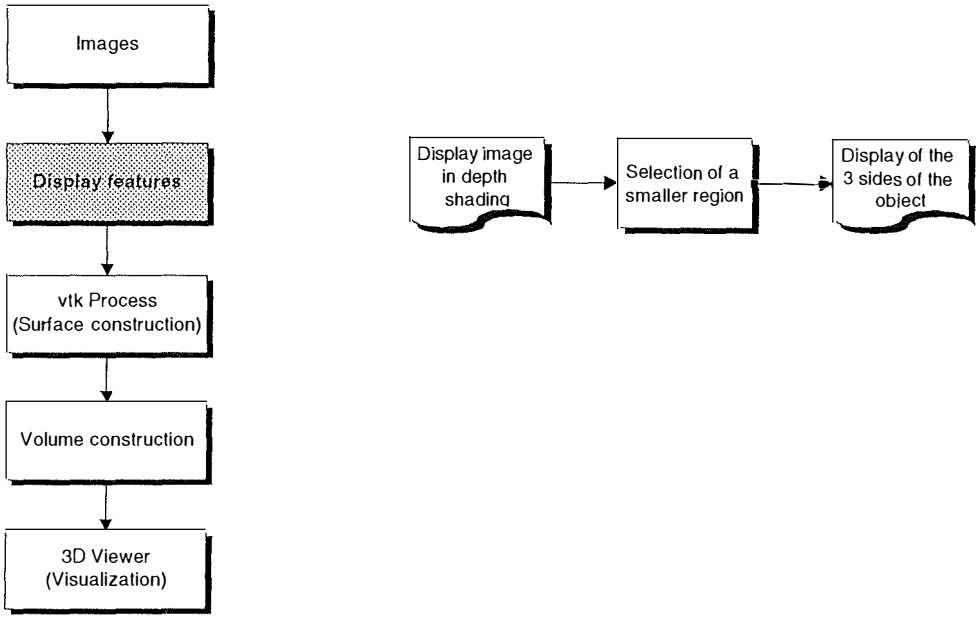


Figure 2-7 : Displaying images in the whole process

3.1 Gray-scale Images

Gray-scale images are images where every pixel value is a gray level, that is, a color between black and white. Although the number of gray-scales is infinite (as well as the number of real numbers between 0 and 1), they are usually encoded in 8-bit or 12-bit values. In the first case, every pixel is stored in one byte and can take 256 different gray levels and in the second one, it can take 4096 different values (usually encoded in 2 bytes). Once an image is being read, depending on how many bit per pixel there are, one or two bytes are read and converted into gray scales. On a common PC, the largest gray scale is 256, so a 12-bit value has to be « shrunk » to 8-bit, with a small loss of precision, as shown in Figure 2-8.

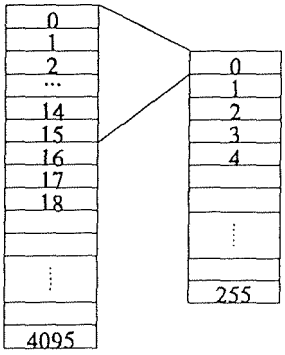
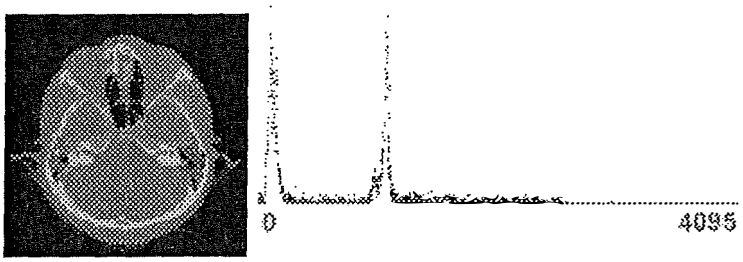


Figure 2-8 : Conversion from 12-bit values to 8-bit values

The conversion is applied for every pixel of the file, and displayed on the screen. This method is used for every gray-scale image, when it is to be shown on the screen, for display purpose. For example, it is used in our program 3D Surface Maker to display CT scan images when an iso-value (cf. Marching Cubes Algorithm) has to be selected using a histogram. A histogram represents the distribution of the gray-scale values in an image, as shown in Figure 2-9.



(a) (b)

Figure 2-9 : A 12-bit converted in 8-bit image[7] and its corresponding histogram

The problem of the iso-value and the histogram will be discussed more precisely in *Marching Cubes Algorithm* section.

3.2 Depth shading

The method explained below fits over sets of 1-bit slices, if they have to be stacked and look like a 3D object. It has been used in our program 3D Surface Maker to display a set of slices read from a binary file, because they have to be displayed in the same time.

Every slice looks like in Figure 2-10. The image is exactly as it is shown, because it is a 1-bit file, with only black and white dots. Most of pixels are black which means that the current bone is very porous.

[7] CT scan of the head, from a set of 93 slices.

Figure 2-10 : A slice of a trabecular bone[8]

To display a set of these slices, and to be sure the resulting image looks like a volume, we use a very simple method called *depth shading*. It is based on a principle stating that far objects are seen darker then close objects. So an easy way to have a set of slices look like a volume, every slice stacked is given a gray-scale value between 0 and 255, 0 being black and 255 being white. Every white pixel of slice *i* will be "painted" with the color c_i corresponding to the following formula :

$c_i = \frac{i}{n} \cdot 255$ where n is the total number of slices.

The result of applying this method to a 50 slice volume is shown in Figure 2-11.

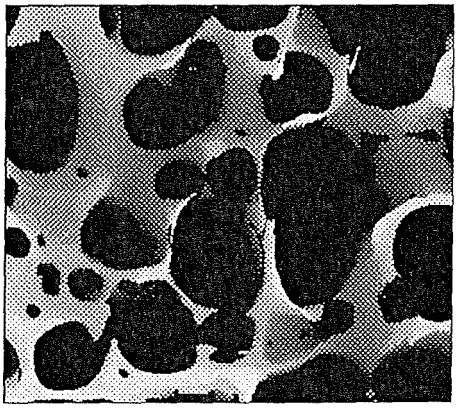


Figure 2-11 : Depth shading method applied to 50 slices

Many other methods can be used instead, but this one was chosen for its simplicity of use. The more slices there are, the more realistic the image looks like. For example, let us compare two images, the first one with 5 slices and the second one with 10 slices. Because there are not enough slices to have "good" shading, one can usually see the different gray levels, as shown in Figure 2-12.

[8] Trabecular bone, section of 7.00 x 6.20 mm

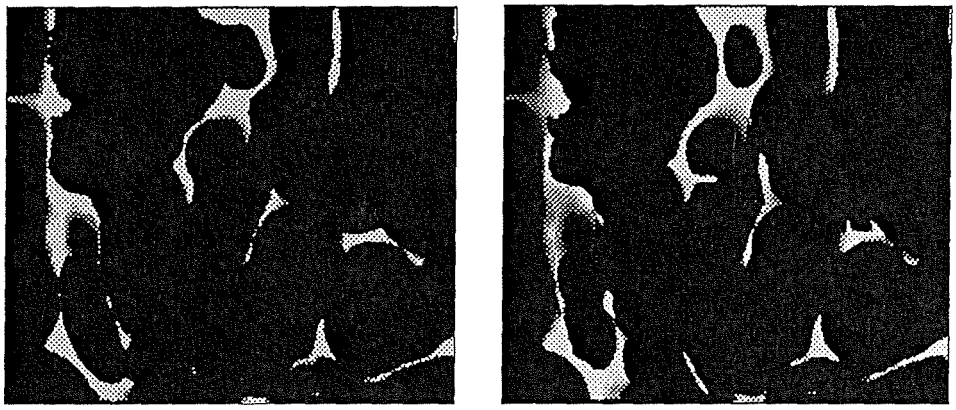


Figure 2-12 : 5 slices (left) and 10 slices (right) depth shading images

3.3 Selecting a region / a subregion

The next step in the process is to select a region in the whole set of slices, because this set can be huge, speaking in terms of number of pixels. For example, the file from which the above images are extracted is 7mm x 6.2mm x 4.02mm, and in term of pixels, 350 x 310 x 201 pixels that is 21,808,500 pixels ! The region can of course be set up by the user for any value ranging in [1, `max_x_pixels`],[1, `max_y_pixels`] and [1, `max_z_pixels`], which allows to select the whole file.

The process of selecting a "good" region is very important, but the depth shading method is not enough, because it does not allow to see all sides of the selected region without a preliminary process. Due to the format of the file, this one could not be read another way to display all sides in the reasonable time, so we use another method to display 3 sides of a selected region, as shown in Figure 2-13 and Figure 2-14. As the selected region is a set of voxels, it can be seen as a volume, where each view is one of its side. The three views shown in Figure 2-14 are the three sides of the cube shown in Figure 2-15.

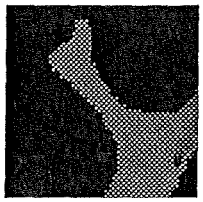


Figure 2-13 : Base image as displayed with simple depth shading

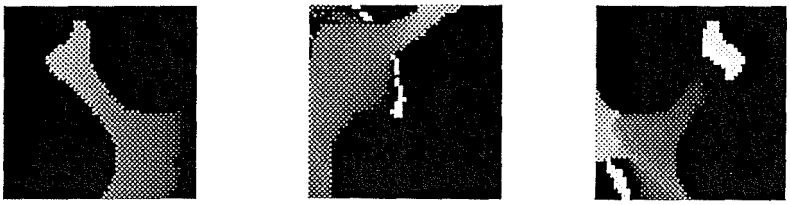


Figure 2-14 : Three other views of the object shown in Figure 2-13

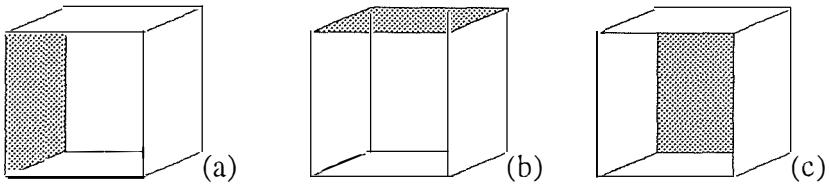


Figure 2-15 : Cube sides

The simplest method to display sides has been used : first, all the pixels for the selected region are read and kept in memory into a matrix, say M_{ijk}. The order to display the main image is explain in the Pseudo-code 2-1:

```
for (k = z_min; k < z_max; k++)
  for (j = y_min; j < y_max; j++)
       for (i = x_max; i > x_min; i--)
            Display (i, j, GetDepthShadedColor(M[z]))
```

Pseudo-code 2-1 : Display the first side of the selected region

The above code displays all the selected region, where pixels with a low z value are displayed first – and so displayed in darker gray levels –, and the greater z value, the lighter gray level.

The way to display any other side of the selected region is to set the parameters (i, j and k) in different order. For example, the left side (Figure 2-15a) is displayed using Pseudo-code 2-2:

```
for (i = x_max; i > x_min; i--)
  for (j = y_min; j < y_max; j++)
       for (k = z_min; k < z_max; k++)
            Display (i, j, GetDepthShadedColor(M[z]))
```

Pseudo-code 2-2 : Display the left side of the selected region

3.4 Importance of 3D visualization

As explained above, depth shaded images give the **illusion of volume** and therefore the user can figure how porous a femur is, when speaking in terms of bones. The fact of giving three different views of an object confirms the opinion the diagnostic maker could have. However, the next step is to allow the user to rotate freely the object, to manipulate it in any direction, to change colors or make a surface translucent in order to really have a good opinion. But this cannot be done with data structured as pixels images are and this is why the next step is to transform the pixels images into a mesh, a net of points connected together. These images are sometimes called vector images because they are made of points in a 3D space and where lines are vectors.

4. VTK Process - Surface construction

The object of this section is to explain how to build a surface made of lines and points – a mesh – out of a set of 2D images. This means a transformation from one representation (a set of pixels) to another one (lines and points in space), and this cannot be done without approximation. In order to explain basics of 3D graphics, we need to give some definitions. This section is divided in three different parts. First, we define some concepts which are commonly used in 3D domain. These definitions are particularly necessary to understand the second part of this section which explains the *Marching Cubes* algorithm. The third part of this section explains the *VTK* toolkit library we used to develop the program.

Figure 2-16 shows the place and contribution of the VTK surface construction in the whole process of visualization.

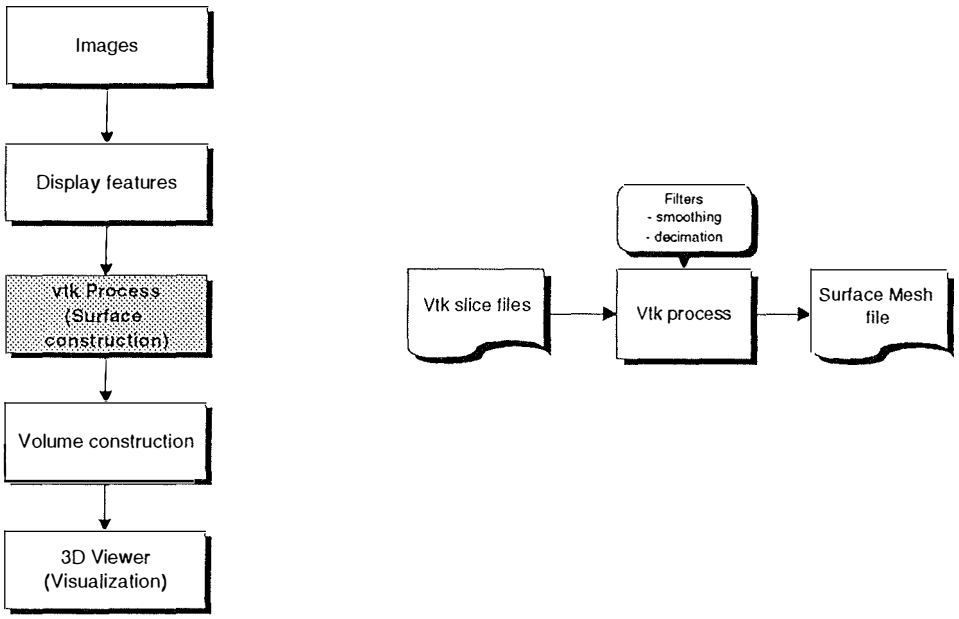


Figure 2-16 : The VTK surface construction in the whole process

4.1 Definitions

There are a lot of useful concepts that could be defined below, however only few of them are necessary to understand basics. A way to develop them is to analyze them from the smallest to the most complex one.

A **vertex** is a single point in a 3D space. According to [SCHROEDER96], it is a 0D cell, used synonymously with point or node. (Figure 2-17a)

A **line** is composed with two linked vertices. (Figure 2-17b)

Three vertices always define one single **triangle** (Figure 2-17c) and four of them, when they do not belong to the same plane, define a **tetrahedron**. (Figure 2-17d)

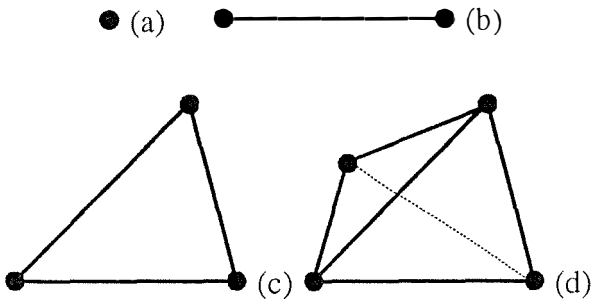


Figure 2-17 : Cell type specification

A **mesh** is a general term for any composition of vertices and lines. An example of a mesh is shown in Figure 2-18.

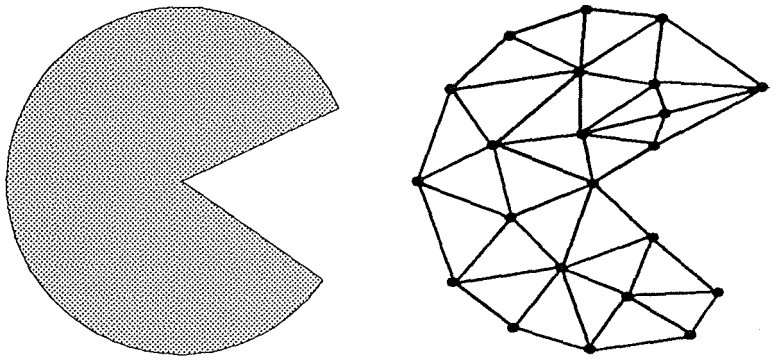


Figure 2-18 : A drawing and its mesh representation (22 nodes)

Figure 2-18 shows a drawing and its mesh representation, after a transformation (usually contouring the drawing). Any drawing, in 2D or in 3D, can be transformed into its mesh representation. The mesh representation is also called the *polygonal representation.* [WATT96] defines the polygonal representation as the classic form in three-dimensional graphics, where an object is represented by a mesh of polygonal facets. In the general case, an object possesses curved surfaces and the facets are an approximation to such a surface (see Figure 2-18 and Figure 2-19)

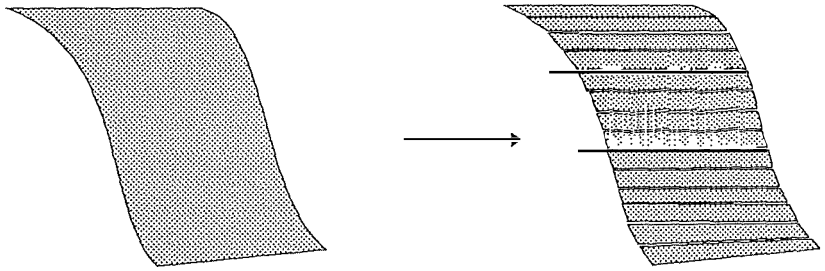


Figure 2-19 : Approximation of a curved surface using polygonal facets [WATT96]

4.2 Marching Cubes Algorithm

4.2.1 Problem

Initially, a set of 2D images represents a 3D volume that can be seen as a parallelepiped full of pixels. An example is shown in Figure 2-4. The problem is the same as in two dimensions. As long as we deal with pixels images, we cannot enlarge or shrink the images without losing precision or piece of data. The idea is to transform the set of pixels into a mesh that can be manipulated easily with mathematical functions while the precision is not changed. Briefly, the

Marching Cube Algorithm [LORENSEN87] builds a surface mesh from a set of following slices depending on an intensity value called the **isovalue**. It uses at least two slices at a time, and try to find a contour for each of them. Finally, it connects both contours which becomes the mesh.

4.2.2 Marching Cubes

The Marching cubes algorithm described below was first explained by W.E. Lorensen in [LORENSEN87]. First, we will detail the algorithm, then we will discuss about the importance of the isovalue and the help brought by the histogram. To ease the understanding of the algorithm, we will first detail the algorithm of *marching squares*, which works for 2D images only.

A. Algorithm

The first step in the marching squares algorithm is to proceed to contouring for each slice. This step is very important because it selects what will be part of the mesh and what will be rejected according to a color value, the **isovalue**. Indeed, when we see a color picture the eyes can easily separate similarly colored areas. The process of contouring works in the same way. Many examples are well known like weather maps where different colors mean different temperatures or topographical maps where different colors mean different elevation in relation to the sea level. In medical imaging, different colors correspond to different body tissues like skin, bones or other organs. We understand better why this process is very important particularly if we want to study bones or skin. The condition is to have the right isovalue. We will discuss that problem in the next paragraph.

Let us consider the 2D grid in Figure 2-20 representing a piece of an image where values are pixel color and let us decide we want to separate values above and below pixel color 5. The problem lies in the fact that pixels color values of 5 are not present everywhere so we sometimes need to interpolate. The easiest method is to linearly interpolate so when the contour must cross an edge where values are 0 and 10 at its two endpoints with a contour line of five, the contour will cross in the midpoint of the edge. Figure 2-20 shows in light gray the contour line for contour value of 5, the dashed line shows the contour line for an isovalue of 4.

Once all the points on the edges are generated, they have to be connected. The marching squares algorithm uses the method "divide and conquer" technique. This method presupposes that each cell can be crossed by a finite number of ways. The different cases are summarized in Figure 2-21 where the dark vertices represent a value above the contour value.

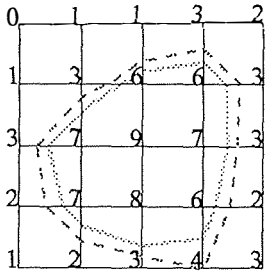


Figure 2-20 : Contouring a 2D image with a isovalue of 5

To better understand the way the Marching Cubes Algorithm works, let us find out how it would work with the example in Figure 2-20. Each intersection of the lines represent a pixel where the color value is shown above. Colors of pixel 0 (left top most) and pixel 1 (next one on the right) are both below the isovalue 5, so they are ignored. Proceeding this way, all pixels in the line are ignored. However, in the second line, between pixels of color 3 and 6, the isovalue of 5 lies near 6, exactly at $2/3^{rd}$ between pixel of value 3 and 6. In the same way, between the two lines, the same isovalue goes between pixels of color 6 and 1, exactly at $4/5^{th}$ between theses pixels. The algorithm runs from one pixel to another one until it has built up the full contour.

The number of cases in Figure 2-21 depends on the number of vertices per cell (a cell is a square made of 4 vertices and 4 lines, like any of the 16 squares in Figure 2-20 or in Figure 2-21) and the number of inside/outside relationships a vertex can have with respect to the contour value. A vertex is considered inside, respectively outside, a contour if its scalar value is larger than, respectively lower than, the value of the contour line. In the present case we have 4 vertices per cell and each vertex can be either inside or outside the contour which gives 2^4 possibilities.

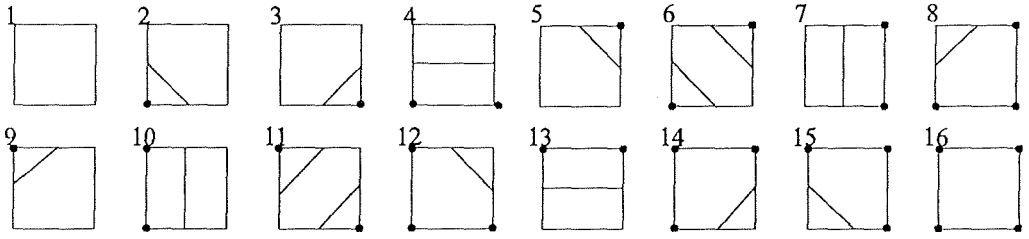


Figure 2-21 : Sixteen different marching squares cases

Each case can be encoded using 4 bits according to the state of every vertex. The algorithm computes for each cell the 4-bit value and uses the 16 cases lookup table to know which case to use. When the right case is selected the location of the contour line is computed by

interpolation. The algorithm processes a cell and then marches to the next one. After all cells are visited, the contour is completed.

We summarize the algorithm in Pseudo code 2-3. [SCHROEDER96]

```
For all cells
  Select a cell
  Calculate the inside/outside state of each vertex
  of the cell
  Create an index by storing the binary state of
  each vertex in a separate bit
  Use the index to look up the topological state of
  the cell in a case table.
  Calculate the contour locations for each edge in
  the case table
```

Pseudo-code 2-3 : Marching squares algorithm

The 3D version of the marching squares is the marching cubes. Instead of 16 cases there are 2^8 cases, i.e. 256 possibilities. However, using rotations and mirroring, we can reduce this number to 15 cases summarized in Figure 2-22.

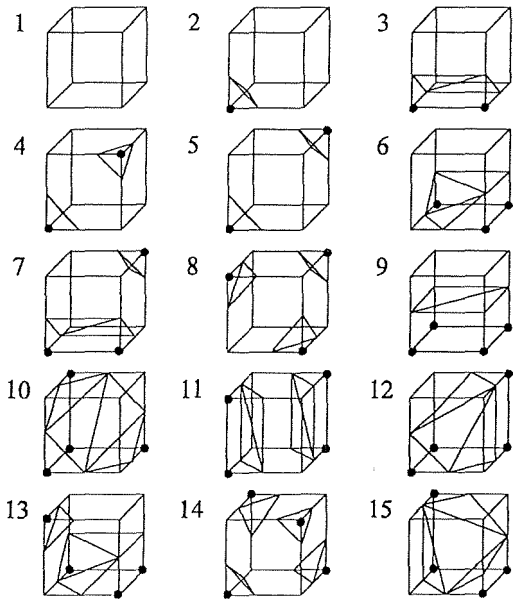


Figure 2-22 : Fifteen cases for marching cubes algorithm

Each cube is the result of the superposition of two slices as shown in Figure 2-23.

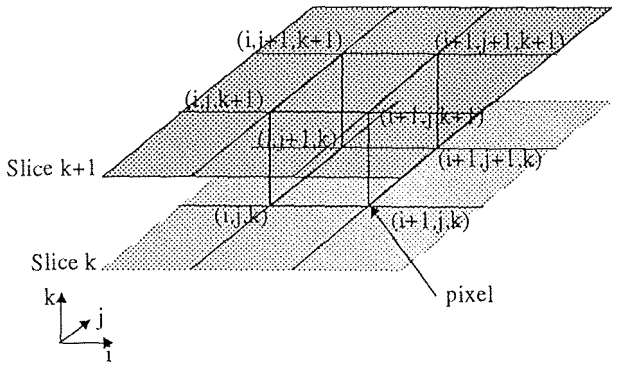


Figure 2-23 : Construction of a cube with two slices

Each slice corresponds to a 2D image like in Figure 2-20. The algorithm processes two slices at a time. It starts with the first two slices and "marches" through the cubes generated by the pixels in the two slices. When all points have been connected, the algorithm takes the third slice and processes it with the second one, and so on for the next ones until the last one. When all the slices have been processed, the surface mesh is ready.

B. ISO Intensity Value

As we have seen in the previous paragraph, the isovalue is very important and two slightly different values can give two very different meshes. The example shown in Figure 2-20 shows the different results with isovalue of 5 and 4. The gray line shows the contour value for an isovalue of 5, the dashed line represents a contour line for an isovalue of 4. We can easily imagine the problem in the medical imaging domain. In 12 bit images where values can range from 0 to 4095 the selection of the right isovalue can be very complex. A good mesh representing an organ or bone depends on the selection of the right isovalue.

C. Histogram

The histogram can help to select a proper isovalue. According to the [AHC86], a histogram is "*a graphic representation of a frequency distribution in which the widths of contiguous vertical bars are proportional to the class widths of the variable and the heights of the bars are proportional to the class frequencies.*" In imaging domain, each class usually corresponds to a color or a range of colors, and the height of each bar represents the number of pixels painted with this color in the image. The histogram can then be useful when it is visually connected to the image it represents. For example, one should be allowed to click with his mouse in the image and the program should show in the histogram the pixel value and so the total number of so-colored pixels. On the other hand, a single click in the histogram should show all pixels corresponding to the selected value in the image. Figure 2-24 shows a screen capture of a dialog box using a histogram.

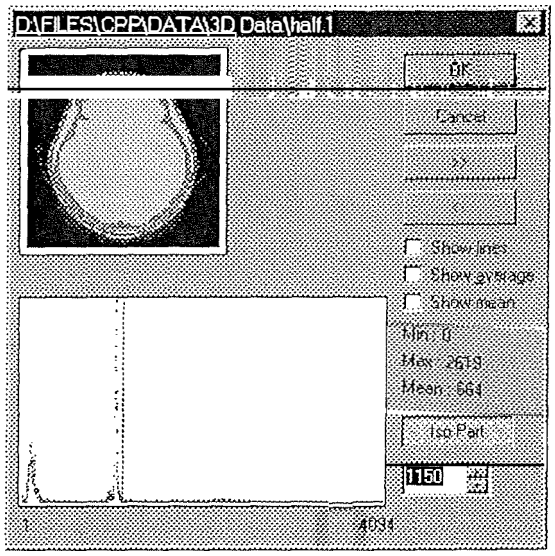
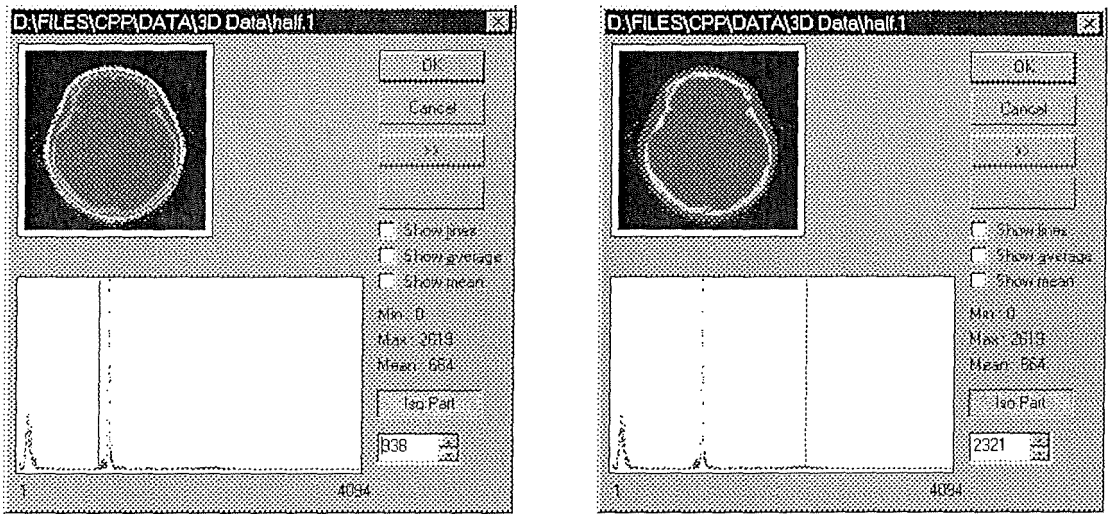


Figure 2-24 : A dialog box with a histogram (from 3D Surface Maker)

The upper left part of the dialog box represents the image, in this case, a CT scan image from a human skull. The bottom left part shows the corresponding histogram, where we can observe three different peaks. The first one starting from the left corresponds to the black part of the image, which is not useful since out of the skull. The program allows to start the histogram from any value in the allowed range, so it is possible to skip this part.
The second peak corresponds to the soft part of the skull, that is, the brain, the blood and other tissues. The third one, which is not very visible, corresponds to the bones and is more spread.
The usefulness of the histogram is straightforward. If we want to apply the marching cubes algorithm in order to retrieve the bones structure of a set of slices, we use the histogram to display the distribution of the colors, then we select the right value corresponding to the bones

(in Figure 2-24, the value 1150 corresponds to the soft tissues, so the algorithm will keep bones) and we run the algorithm with the selected isovalue. This is particularly interesting because two set of slices may have different values for extracting bones. Figure 2-25 shows the selected regions for an isovalue in the peak corresponding to the soft tissues and for an isovalue in the peak corresponding to the bones.



(a) (b)

Figure 2-25 : Skin (a) and bones (b) selected regions

In the images at the upper left part of the dialog boxes, the light grayed parts corresponds to the pixel values ranging from -10% to +10% around the isovalue. In the 3D Surface Maker program, these regions appear in yellow.

4.2.3 A whole example

This paragraph summarizes the current section. The below example is based on a slice set from [SCHROEDER96] and all images have been made from 3D Surface Maker, the program we wrote.

Slice set

Each slice is part of a set of 93 slices taken from a child skull, from the neck to the middle of forehead every 1.5 mm. The following image (Figure 2-26) is an excerpt of this set and was taken perpendicularly to the spine approximately through the middle of the nose.

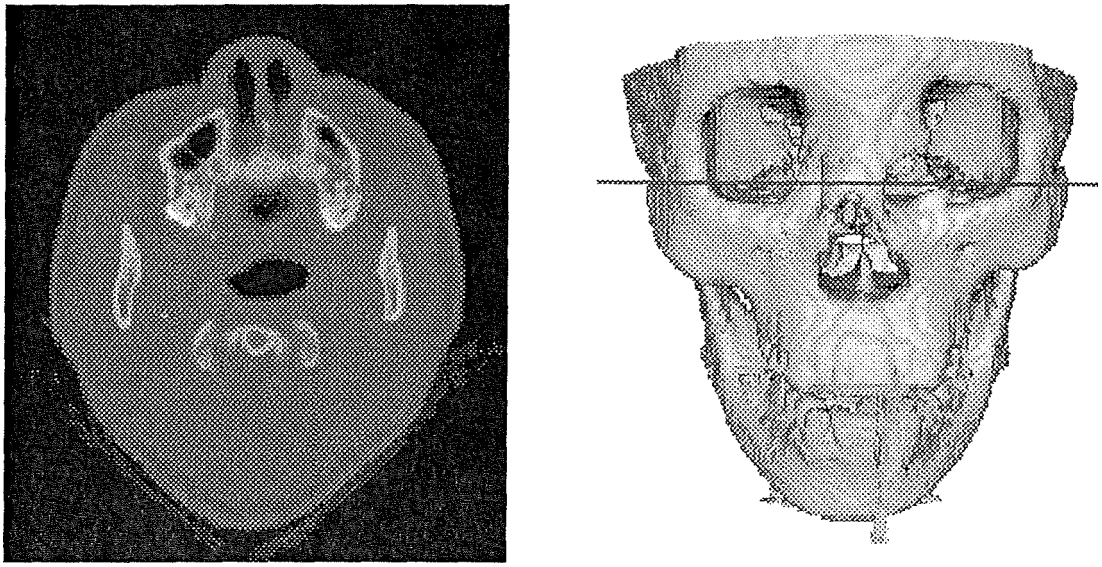


Figure 2-26 : CT slice through a human head and its corresponding position

Contouring

As explained above, each slice is contoured and then the algorithm is performed between several slices. The contouring for the previous slice is shown in Figure 2-27a.

(a) (b)

Figure 2-27 : Contouring a CT scan and contouring bones only

According to the isovalue, only a small part of the contours will remain in the next step. If we want to retrieve bones out of the image, we get the result shown in Figure 2-27b.

Mesh

The connections between each image are then produced thanks to the Marching Cubes algorithm, as shown in Figure 2-28 for two slices (Slice 45 and slice 46).

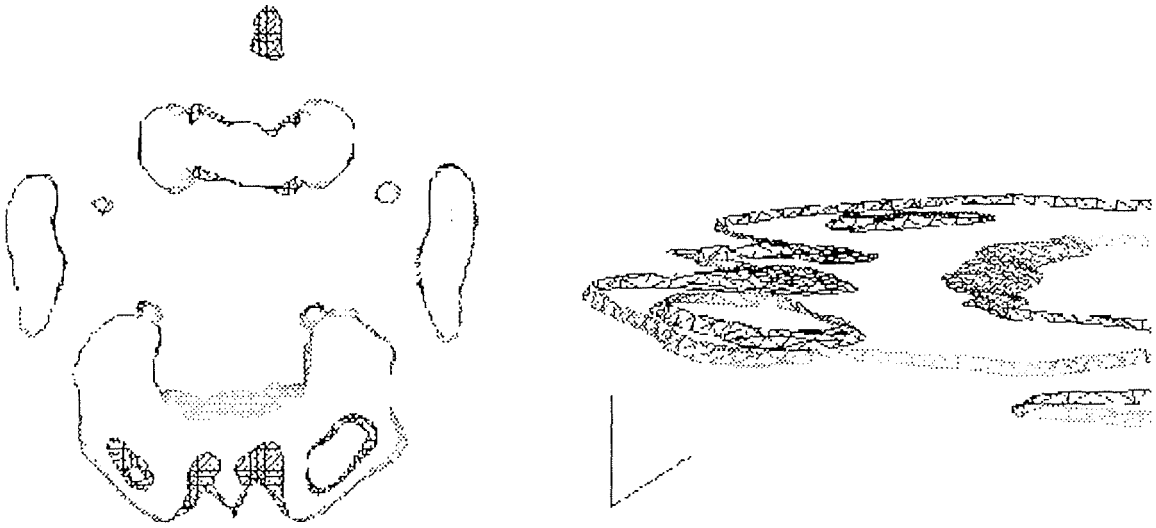


Figure 2-28 : Marching cubes applied to two slices. (a) Top view. (b) Side view

One can observe the triangles defining the surface created between the two slices.
The whole 3D surface is presented in Figure 2-29. Left image is the mesh surface and the right one is the rendered surface (see the below Rendering section for more details).

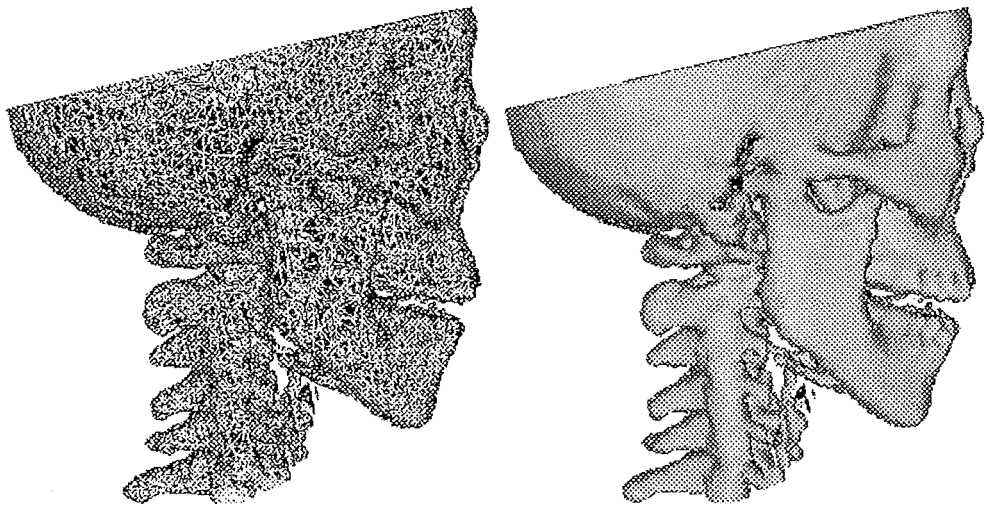


Figure 2-29 : Mesh and rendered surfaces

4.3 The Visualization Toolkit

This section will describe one toolbox that can be used to develop graphical and imaging software. There are many toolboxes available but we decided to use the Visualization Toolkit [SCHROEDER96] for several reasons. First, this library is free and its code as well as documentation are available from anywhere on the Web[9]. Secondly, the toolkit is platform independent so a program written on a PC can also run on a UNIX-based machine[10]. This is particularly interesting when developing medical imaging software because UNIX machines such as Silicon Graphics workstations are much more powerful speaking in terms of processor as well as graphical devices.

The *VTK* toolkit is also written in object oriented code C++ which is very fast and worldwide spread. The library itself is object oriented written, with abstract classes and derived classes organized in three categories : common library, graphics library and imaging library. The common library handles basic operations and objects, like Vertices, Points, Polygons, but also the Marching cubes algorithm, the Bitmaps and mathematical functions. The graphics library handles graphical transformations, filters, cameras, lighting, ... Finally, the imaging library is a new library introduced in 1997 particularly designed to help in image processing. We will describe the object oriented of the *VTK* toolbox and the rendering process.

4.3.1 Object oriented

As we already said, the visualization toolkit which we will call VTK from this time onwards is object oriented. Below we will briefly describe the objects structure.

A. Overview

Object oriented software systems are more and more chosen when deciding for a development design. They are more modular, easier to maintain and to upgrade and easier to explain and to understand. On the other hand, they require more rigor and methodology but these are qualities needed when developing complex programs. Visualization is an example of a complex program, ever evolving which requires never-ending modifications of the existing system. As

[9] http://www.cs.rpi.edu/~martink

[10] This is true as long as the program does not call any proprietary functions. For example, VTK library supports basic functions to handle windows but these methods no longer work when they are migrated to another system. Especially, high level functions such as Microsoft Foundation Classes provided with Visual C++ are not working with X-Window and consequently is not compatible. However, it is possible to develop a fully functional program working on both systems.

[SCHROEDER96] says, "*a good software design should be robust, understandable, extendible, modular, maintainable and reusable*".

We can explain these terms as follow : [SCHROEDER96] and [DUBOIS96]

- A **robust system** would handle exceptional conditions and would behave as expected even when used under different circumstances, an **understandable system** would allow anyone different from the implementor to use the program. This means the program should be logical and follow ergonomic rules for example.
- An **extendible system** is a system carrying out old tasks while accepting to perform new ones. This should be done without too many changes in the existing code since the more changes are made, the more errors are introduced.
- A **modular system** means functions without or with few relationships are not gathered in same modules and therefore minimizes changes in different modules. They should share name conventions and protocoles.
- A **maintainable system** should be thought while designing and developing a software.
- The **reusability** of programs reduces the work of adding new features whereas the design of such a reusable software is usually difficult and takes extra time.

"*In computer Science, an object is an abstract entity that embodies the characteristics of a real-world object. [...] Objects are the result of a programming methodology rather than a language*" [FAISON94]

These **objects** encapsulate **properties** and **behavior** of the entities within a system. Each object has an identity which distinguishes it from other objects. The major difference between conventional and object-oriented system is the way they approach data abstraction. Conventional systems limit abstraction to data typing where object-oriented system create abstraction for both data typing and methods applied to the data. **Inheritance** is a mechanism that eases the addition of new classes when these classes are very similar. This involves a hierarchical classification of the system we want to develop.

The *VTK* can be described with the Object Modeling Technique (OMT) developed at General Electric and explained in [RUMBAUGH91]. This model uses three models to specify an object-oriented design: an object model, a dynamic model and a functional model. These three models are described below and an example is given for each model.

B. The object model

The object model identifies each object in the system, shows its properties and the reltationships with other objects. Each object is represented with a rectangle where its name is

at the top, then all the attributes are listed. Finally all methods are shown. Relations between objects are called associations and are shown with lines connecting relating objects. Like in the Entity-Relation Model, relations can have different cardinalities[11] and association can be labeled with roles[12] [BODART93]. Appendix 1 shows the Object Model for *VTK*.

C. The dynamic model

Where the object model describes the static part of a system, the dynamic model details the sequence of events and time dependencies of the system. This is specially useful to design control system and user interfaces where dialog boxes follow other ones in a particular order. The way the visualization toolkit was written has limited sequence and control aspects, however, a visualization program, using *VTK* or any other graphics toolbox is mainly based on this model, as it will be discussed later. Figure 2-30 shows a basic example of a visualization program.

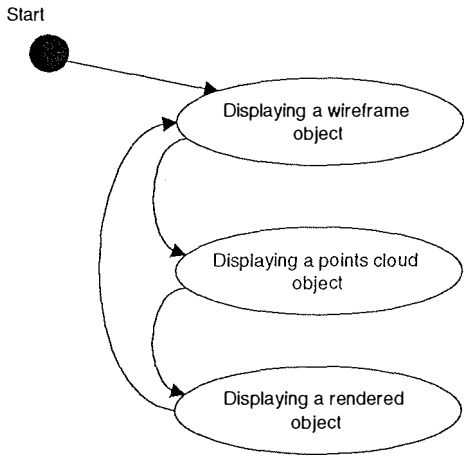


Figure 2-30 : Dynamic Model applied to a basic visualization program

Each oval in the diagram represent a state where an arrow shows a transition from one state to another. Once the program starts, the display type is set at "wireframe". If one changes this type of display, the object is displayed as a points cloud. If the display type is changed again, the object will be shown as a rendered surface. Changing the display type once more would show the object as in the first step.

D. The functional model

The functional model shows flows in the system, how data move through the system. The graphical representation of the functional model is called the data flow diagram. Its major

[11] One-to-one, one-to-many and many-to-many.

[12] Roles are names given to associations and are used to further describe the nature of the association

components are data sources and processes. Data sources are shown by rectangles and processes are represented by ellipses. The visualization process of the programs we wrote is presented in Figure 2-31 in the data flow diagram representation. It has been detailed in the Visualization Process section. One can observe the different data representations, slices, triangles, polygons and pixels and the different two ways to display the objects, either the surface or the volume objects.

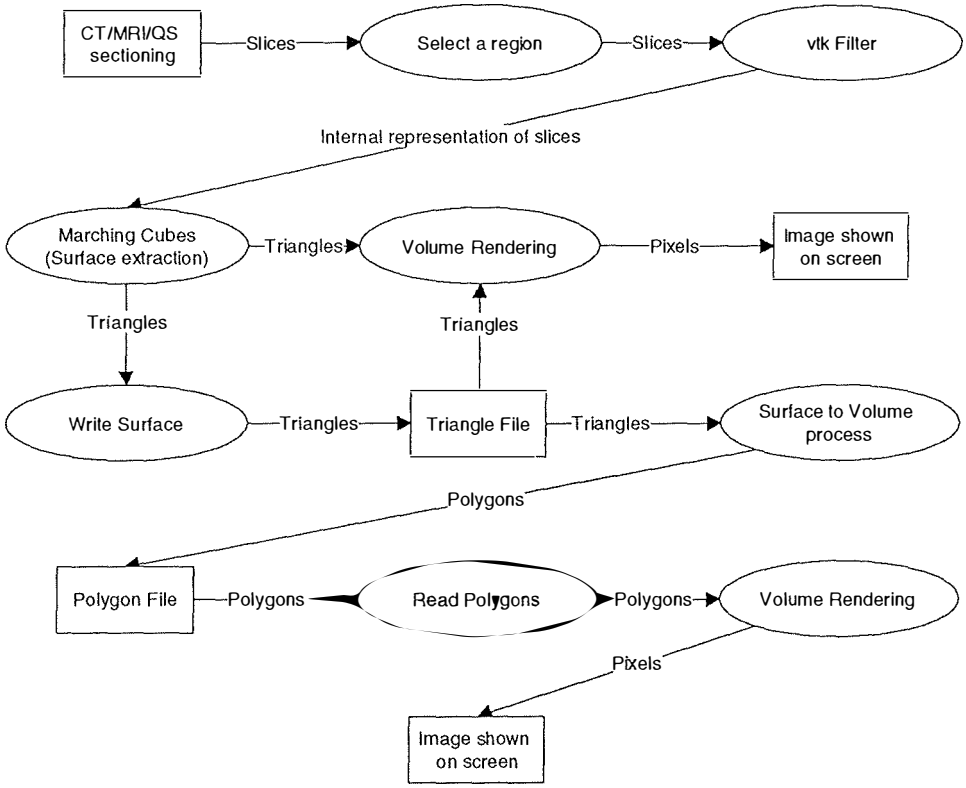


Figure 2-31 : Data flow diagram for the visualization process

4.3.2 Volume rendering with *VTK*

The process of generating images with computers is called *rendering* [SCHROEDER96]. [WATT93] defines *rendering as the collection of operations necessary to project a view of an object or a scene onto a view surface.* Indeed, the main problem with 3D graphics is that we want to display a 3D object onto a 2D surface screen. There are many types of rendering ranging from simple depth shading method (see above) to sophisticated 3D techniques.

5. Volume construction

Figure 2-32 shows the place of the volume construction in the whole process.

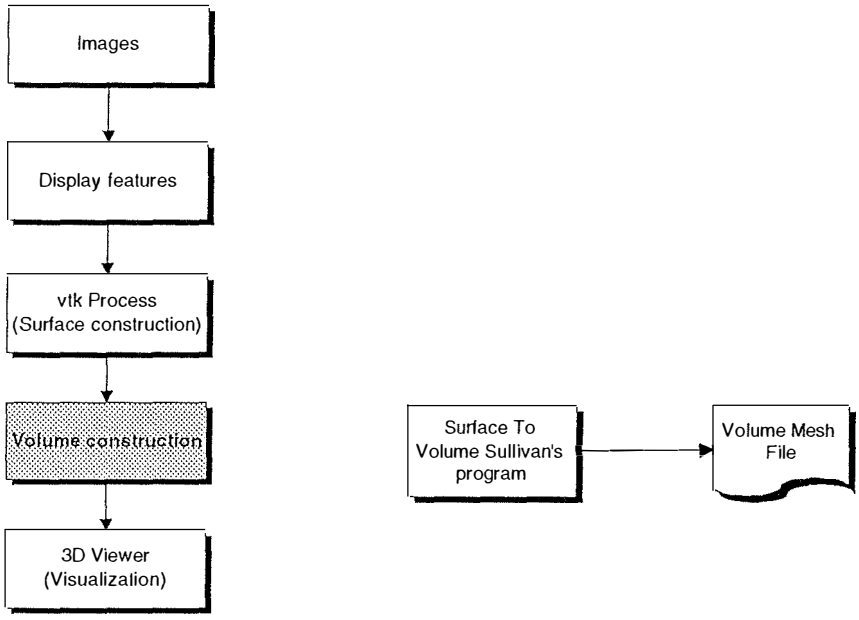


Figure 2-32 : Volume mesh construction in the whole process

5.1 Overview

Once the surface has been built, and in order to use numerical analysis, we apply another algorithm written by [SULLIVAN95]. This algorithm takes place in a command-line program called **3dmesh** which runs under UNIX. This is one of the main reasons for which we had to develop 3D Viewer. Indeed 3D Viewer allows files created with 3D Surface Maker (using Marching Cubes Algorithm) and the ones created with 3dmesh to be displayed in the same time and in the same scene[13]. This operation allows to compare results of 3dmesh with its inputs, since there is no other validation process to check results from 3dmesh. This section describes briefly the program *3dmesh* and its main functions.

The mesh generation method is divided into several steps. The user enters the boundary geometry, composed of line segments, at the desired boundary resolution. Nodes are deployed

[13] See below : *The scene and the planes* in *3D Viewer - A visualization program*

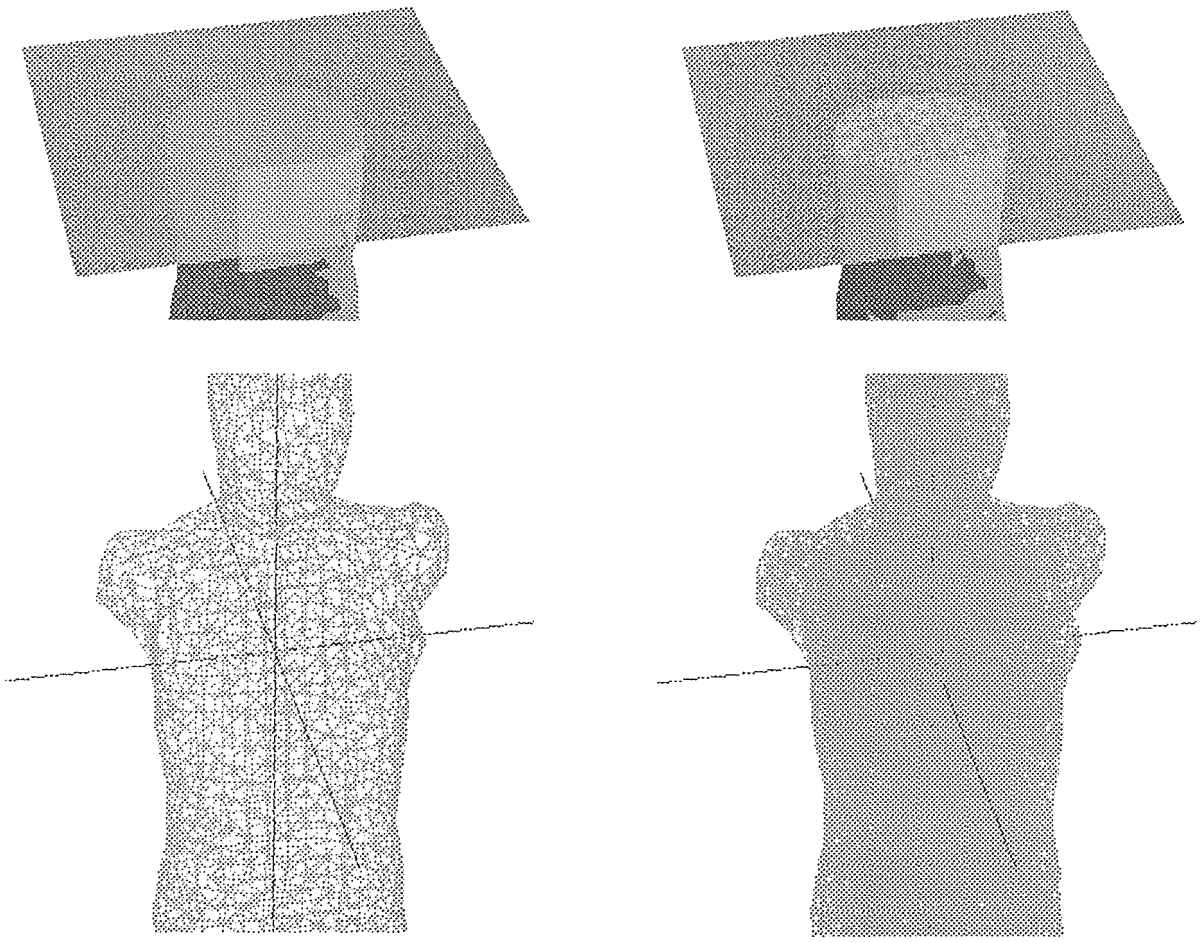
in the domain by offsetting the initial array of nodes located on the boundary inside inward along vectors normal to the boundary geometry and processing the resulting new points to determine new nodal locations. These new nodal locations are offset initiating another cycle. This sequence of offset-process-offset continues until nodes are deployed throughout the domain.

Finally, an element connection via Delaunay triangulation is applied to produce the final mesh. The term *layer* is used to describe a set of points that are related by their similar depth toward the interior of the domain from the boundary. The term *points* is used to refer to the discrete offset locations in a layer before the locations are processed to determine their final locations. *Nodes* are the final, processed locations that make up the **actual mesh**. A *row* is represented by line segments which connect the points in a layer to form a continuous closed loop(s). The layer of nodes upon which the offsetting process is currently performed is the *active layer* and the resulting offset point locations is called a *new layer*. The active layer is referred to as the parent of the new layer. A complete cycle of offsetting an active layer to create a new layer and processing the new layer to determine nodal locations is referred to as a *step*. At the end of each step the new layer is ready to become the parent layer for the next step. Portions of the domain in which nodes have been deployed are referred to as *meshed* regions. Those regions in which no nodes have been deployed are *unmeshed*.

Initially, the unmeshed domain consists of the entire region within the geometric boundary. Following an offset step, the new layer of nodal locations forms a new interior boundary surrounding a new domain of unmeshed space. This new boundary layer, and the unmeshed domain are conceptually and structurally identical to the original boundary and domain and may be passed trough the offset process again to form yet another layer of nodes. The process is repeated in sequence until the shape of the new offset layer converges in the interior and the domain is filled with nodes. During offsetting, the shape of the new offset layer will not be the same as that of its parent layer or of the original boundary. The mesh density function gives local control over the deployment. Additionally, offsetting along vectors normal to curved boundaries causes convergence in concave regions and divergence in convex regions. These factors coupled with loop intersections change the shape of the working layer.
A sequence of tests which increase in computational complexity are used during the offsetting process. Neighbor distance tolerance checking is followed by an angle tolerance check with a loop intersection check performed thereafter. This test sequence rectifies each new layer to ensure an appropriate node deployment in the domain. [SULLIVAN95]

5.2 Examples

The images shown below are exerpt from our program, 3D Viewer.



(a) (b)

Figure 2-33 : A surface mesh (a) and a volume mesh from 3dmesh (b)

6. 3D Viewer - A visualization program

3D graphics are two-dimensional images on a flat computer screen that provide the *illusion* of depth, the third dimension. The object of this chapter is not to define all 3D concepts in order to allow the reader to manipulate them or to develop a 3D program but to be able to understand the main concepts used in this thesis. In the critical application of the TRIDENT methodology in the next chapter these concepts are used a lot, that is the reason why we think an overview can be useful. This chapter is divided in three parts: Main principles section – or

3D basics – describes the main parts in 3D graphics, the scene, lights, colors, ... The second section will describe the architecture of the visualization program.

Figure 2-34 shows the visualization process and contribution of 3D Viewer in this process.

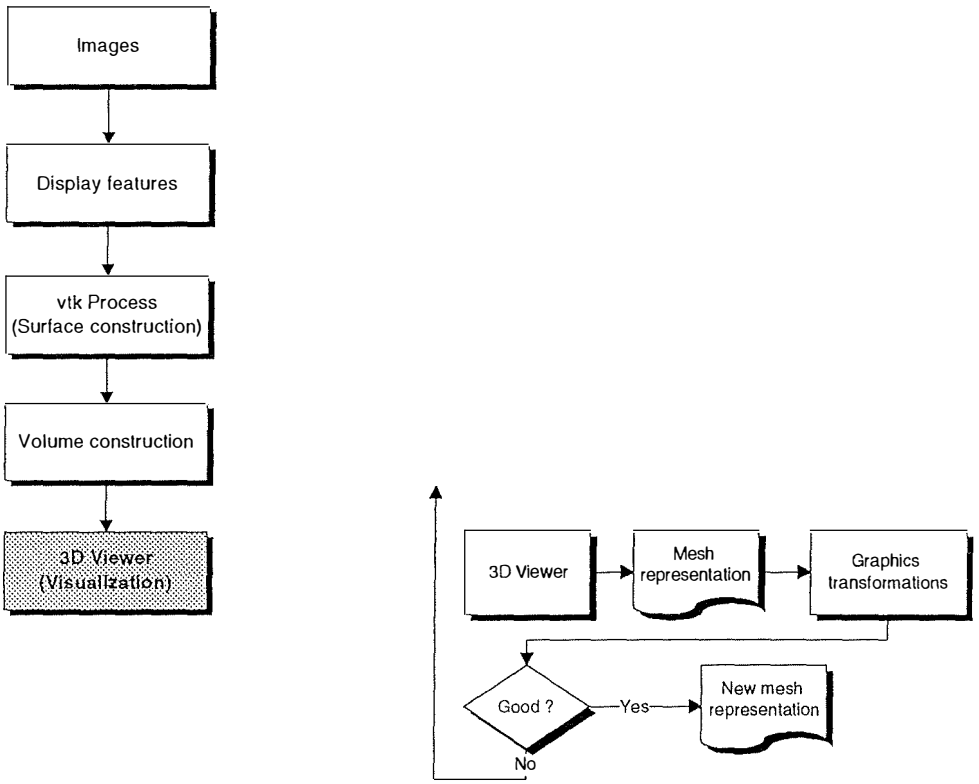


Figure 2-34 : 3D Viewer and its position in the whole visualization process

6.1 Main principles

This section intends to explain the main components of a basic 3D program. Indeed, every so-called 3D program must include a scene, should include geometric transformations such as rotations or scaling, colors and lights. We will also talk about views and cutting planes which seem to be very important in visualization problems.

6.1.1 The scene and the planes

The scene is the first step when building a 3D graphics program. It is where the objects are displayed and usually physically corresponds to the screen or the window in a multiple-window environment. Besides the objects contained in the scene, the latter contains two important objects called far plane and near plane, as shown in Figure 2-35.

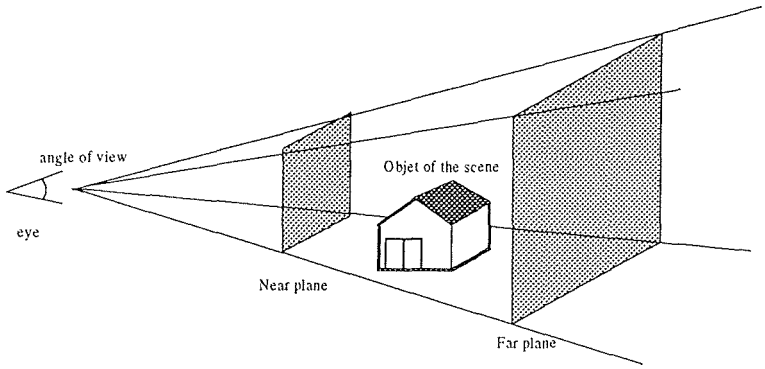


Figure 2-35 : A scene within its near and far planes

Near plane

Near plane is the plane situated between the position of the viewer (i.e. the eye) and the scene, and any object that is situated between the near plane and the eye is not shown on the screen.

Far plane

Far plane is the plane situated at the end of the scene and from which any object situated further is not shown.

6.1.2 The geometric transformations

Geometric transformations are useful if modifications on the objects are needed. For example, the object shown on the screen can hide something behind itself and we may want to rotate it to check up. The most common transformations in 3D graphics are Rotation, Scaling and Translation. Since 3D graphics are made of vertices, and each vertex is made of 3 coordinate points, the obvious way to handle these transformations is to work with matrices. These transformations are summarized in Figure 2-36.

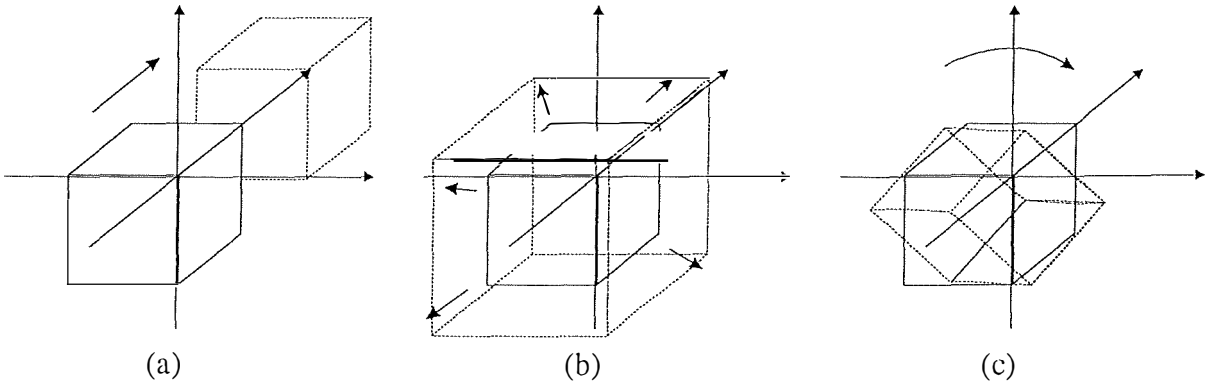


Figure 2-36 : Translation (a), Scaling (b) and Rotation (c) transformations

Transformation matrices [SCHROEDER96], [WATT93] and [WRIGHT96]

Using matrix notations, a point P is transformed under

- translation as P' = P + T (2-1)
- scaling as P' = PS
- rotation as P' = PR

where T, S and R are respectively a vector of translation, a matrix of scale factor and a matrix of rotation. This works fine as long as we are working with simple 3D objects without any perspective considerations. Indeed, to give perspective effects, we need to add a fourth element. So where a Cartesian point is defined as P = (x, y, z), a point represented in its *homogeneous coordinates* is defined by a four element vector (x, y, z, w) [SCHROEDER96] [WATT93]. Therefore, any transformation matrix will be 4×4 matrix.

The conversion between Cartesian coordinates and homogeneous coordinates is given by (2-2)

$$x = \frac{x_h}{w_h} \qquad y = \frac{y_h}{w_h} \qquad z = \frac{z_h}{w_h} \tag{2-2}$$

Object translation

Translating an object can be useful when one wants to move it from one place to another, for example to superpose objects or on the other side to show two superposed objects. To create a transformation matrix that translates a point (x, y, z) in Cartesian space by the vector (t_x, t_y, t_z) we need to build the translation matrix

$$T_T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-3}$$

and then multiply it with the homogeneous coordinate (x_h, y_h, z_h, w_h). We don't need to set a value for w, which controls perspective aspect but we can set it at a value of 1 (no perspective effect).

The translation of only one or two axis is possible when setting the other values to 0. For example, to translate on the X axis only, t_y and t_z are worth 0 in the matrix T_T shown in (2-3) The translated point (x', y', z') in homogeneous coordinates is obtained as shown in (2-4).

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2-4}$$

Using (2-2) to get the Cartesian coordinates, we have the right result :

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \\ z' &= z + t_z \end{aligned} \tag{2-5}$$

Object scaling

Scaling an object can be useful in two cases. First of all, it has the same effect as zooming since one can shrink or enlarge the object to fit its needs. Secondly, scaling can be useful when working with multiple objects and these ones are not the same size or the same scale.

Let's take a point (x, y, z) and apply a scaling factor of (s_x, s_y, s_z). The transformation matrix is

$$T_S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-6}$$

Just like with translation, the new point is obtained as shown in (2-7)

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2-7}$$

and using (2-2), the Cartesian coordinates are :

$$\begin{aligned} x' &= s_x x \\ y' &= s_y y \\ z' &= s_z z \end{aligned} \tag{2-8}$$

Note that scaling only on one or two axis is also possible by setting s_i to value 1.

Rotation of an object

Rotating an object can be useful to check all sides of the object. Since rotation is a little bit more complicated than the other two transformation aboves, we will describe rotation around

one axis at a time.

To rotate an object around the X axis by angle θ we use the matrix T_{R_x} :

$$T_{R_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-9}$$

T_{R_y} and T_{R_z} rotation matrices are shown in (2-10)

$$T_{R_y} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T_{R_z} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-10}$$

The matrices in (2-10) correspond to the three equations systems shown in (2-11a,b and c)

$$\begin{cases} x' = x \\ y' = \cos\theta + z\sin\theta \\ z' = -y\sin\theta + z\cos\theta \end{cases} \tag{2-11a}$$

$$\begin{cases} x' = x\cos\theta - z\sin\theta \\ y' = y \\ z' = x\sin\theta + z\cos\theta \end{cases} \tag{2-11b}$$

$$\begin{cases} x' = x\cos\theta + y\sin\theta \\ y' = x\sin\theta - y\cos\theta \\ z' = z \end{cases} \tag{2-11c}$$

The global transformation matrix corresponds to a transformation of the axis x-y-z to a new axis x'-y'-z'. We assume the axis x makes the angles ($\theta_{x'x}$, $\theta_{x'y}$, $\theta_{x'z}$) around the axis x-y-z. Similarly, the y' axis makes angles ($\theta_{y'x}$, $\theta_{y'y}$, $\theta_{y'z}$) and z' makes the angles ($\theta_{z'x}$, $\theta_{z'y}$, $\theta_{z'z}$).

We finally obtain the following matrix :

$$T_R = \begin{bmatrix} \cos\theta_{x'x} & \cos\theta_{x'y} & \cos\theta_{x'z} & 0 \\ \cos\theta_{y'x} & \cos\theta_{y'y} & \cos\theta_{y'z} & 0 \\ \cos\theta_{z'x} & \cos\theta_{z'y} & \cos\theta_{z'z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-12}$$

Combinations of transformations

The previous transformations (scaling, translation and rotation) can be combined to represent all type of transformations simultaneously. For example, a translation matrix M_1 applied to a point and then a rotation matrix M_2 applied to the result is represented in (2-13)

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ w'' \end{bmatrix} = \begin{bmatrix} \cos\theta_{x'x} & \cos\theta_{x'y} & \cos\theta_{x'z} & 0 \\ \cos\theta_{y'x} & \cos\theta_{y'y} & \cos\theta_{y'z} & 0 \\ \cos\theta_{z'x} & \cos\theta_{z'y} & \cos\theta_{z'z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} \quad (2\text{-}13)$$

or in the algebraic form :

$$X' = T \cdot X \quad (2\text{-}14)$$
$$X'' = R \cdot X'$$

With matrix multiplication, we can have it simplified as shown in (2-15)

$$X'' = R \cdot T \cdot X' \quad (2\text{-}15)$$

However the user has to keep in mind the computational aspect of matrix multiplication ($O(n^2)$) for every point to transform !

6.1.3 The (objects) color

Color has now all its importance. All computers are sold with high quality 16 million color monitors and nobody could work anymore with a black and white word processor or spreadsheet. It is even more true when talking about computer graphics.

A. Importance of color in visualization

Visualization is based on showing, displaying objects on a screen in order to help to understand problems, to get quick idea or to support a decision. A quick look into an array of numbers is easier when negative numbers are shown in red for example. In medical imaging, bones and soft tissues are more easily distinguished when bones are shown in one color (e.g. gray) and soft tissues in another. This is one reason for which we decided to allow to change color for any object in the program, i.e. any object in the scene, any axis or any bounding box.

B. Components of the color

Color is a wavelength of light that is visible to the human eye.[WRIGHT96]. Wavelengths of

visible light range from 390 nanometers for violet to 720 nanometers for red, this range being usually called the *spectrum*. Figure 2-37 shows the light spectrum ranging from violet to red going through blue, green and yellow.

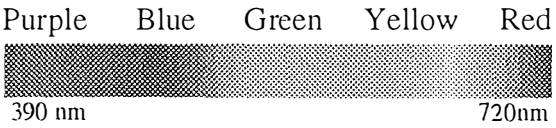


Figure 2-37 : Light spectrum

Computers use two different methods in coding color. The first one is called RGB and represents colors based on their red, green and blue intensities. It can be seen as a 3 dimensional space with red, green and blue axes. The second method is called HSV for hue, saturation and value (brightness). We used the RGB model because it is implemented in most of programming languages.

6.1.4 Lights

Lights are very important in a scene, when displaying an object. In fact, this is probably the most important thing in the process of displaying an object. Indeed, if there is no light, the object will be black, and as [SCHROEDER96] says, this object will be rather *uninformative*. [SCHROEDER96] and [WRIGHT96] agree that the reason that lights are so important is that the interaction between the emitted light and the surfaces of the objects defines what we see. Beside colors, lights bring other effects such as shininess or shading and make the objects look more realistic. When dealing with lights in a program like in 3D Viewer, one has to work with a lot of parameters which are not always easy to tune. The light that illuminates an object is often composed of different light components, the ambient, the diffuse and the specular ones. Once the type of light has been chosen, the right position has to be set, in the three-dimensional space

A. Parameters of the light

1. The ambient parameter

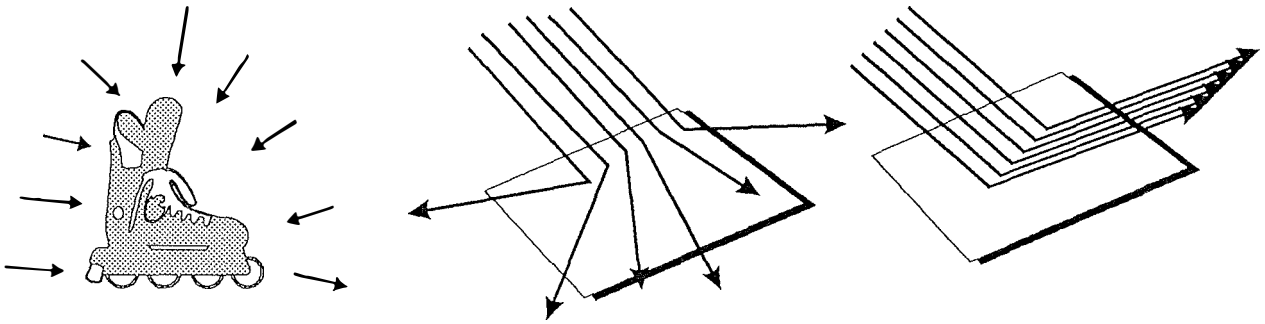
[WRIGHT96] defines an ambient light as a light that does not come from any particular direction. It comes from a source, e.g. a bulb, but the rays have bounced so many around the room that they become *directionless*. A consequence of this is that objects hit by ambient light are evenly lit on all surfaces.

2. The diffuse parameter

A diffuse light comes from a particular direction but is reflected evenly off a surface [WRIGHT96], but the surface will be brighter if the angle made by the light rays with the object is closer to 90° than to 0°.

3. The specular parameter

Specular light is directional as diffuse light, but is reflected sharply and in a particular direction. A specular light tends to cause a bright spot on the surface and rays are parallel. Common examples are the sun or a laser beam. Figure 2-38 shows examples of ambient light, diffuse light and specular light.



(a) (b) (c)

Figure 2-38: Ambient (a), diffuse (b) and specular (c) lights

The effect of specular light is shown in Figure 2-39, where the object, a body trunk, is displayed in 3D Viewer interface. The first image corresponds to all lights off, the second one, to one light *on*, where its specular component is set to 0, the third one, the specular component set to 128 and the last one, the specular component is set to 255 (maximum value). Usually, light components are defined in RGB components, however, we decided to define the specular component in gray scales, because the specular component acts like a shininess component. Specular, diffuse and ambient parameters are components of lights but can appear in different values. Therefore, a light can be composed of only one or two components.

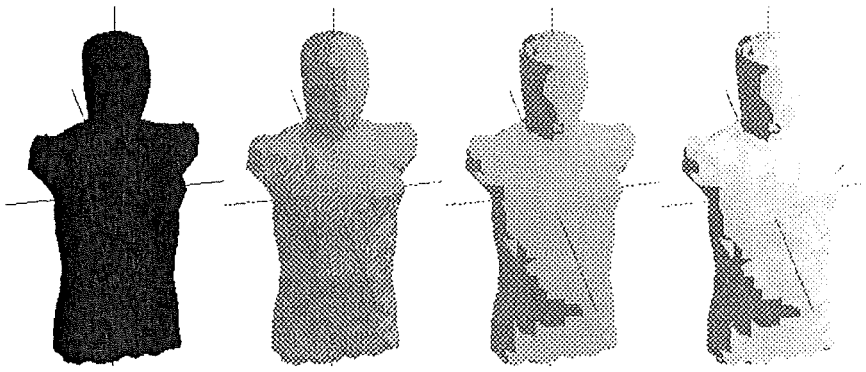


Figure 2-39: Working with specular light

4. The position of a light

The light position can be important when showing a complex object where parts of it are shadowed by other parts of the object. The position of a light is a coordinate in 3D space, i.e. (x_l, y_l, z_l). However, an effect can be given if one wants to give the impression of a light coming from infinite, like the sun. The *far* parameter creates a light where all rays are parallel but where the direction has to be specified. Figure 2-40 shows the same object where the light position has changed. The position of the light is represented by a black dot. The new position of the light allows the user to see the right part of the human body face.

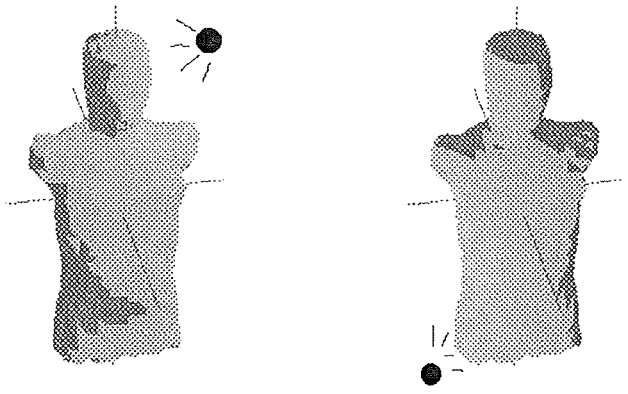


Figure 2-40 : Light positions

B. Material properties

As [WRIGHT96] explains, lights are only part of the equation. Objects have a color, and an object color means that this object reflects this color and absorbs the others. For example, a red car reflects most of the red component of the light, and absorbs most of the others. In most of cases, a white light hits the objects and therefore they appear in their "real" color, but a

blue ball in a dark room illuminated with a yellow light would appear in black because the ball would absorb the yellow component. So objects have also properties like lights. As [WRIGHT96] says, we have to talk in terms of reflective properties for ambient, diffuse and specular light sources. Determining the specular aspect of an object amounts to determine how the object reflects the specular component of the light. Likewise, determining the ambient or the diffuse aspect of an object amounts to determine how the ambient or diffuse components of the light are reflected by the object. [WRIGHT96] suggests however to consider ambient and specular components being equal.

So when defining an object we have to specify its "reaction" to lights. For example, setting the shininess of an object will lead the object to reflect more or less light. The transparency can be a very useful tool when working with a composite object. For example, making the skin transparent can help visualize bones through the skin while keeping an eye on it, in order to see external damage.

C. Surface Normal

When the 3D representation of a piece of trabecular bone is visualized as a surface made up of plain triangles, the notion of **normal to a surface** is important if lighting is used. The normal to a surface determined by 3 vertices of a triangle helps to determine the angle of reflexion of a light ray reflecting on the surface at one point. (Figure 2-41)

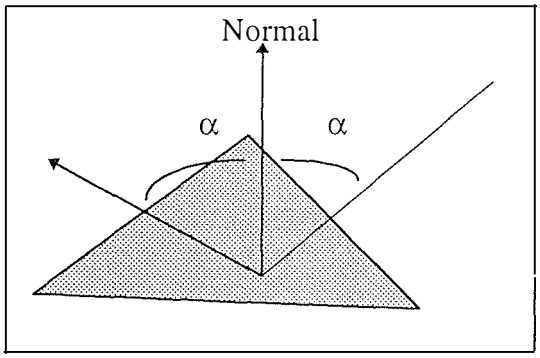


Figure 2-41 : The surface normal

The normal to a surface is a vector perpendicular to the surface and having a direction. When a light ray is touching the surface at one point, with an angle α with the normal, the light ray is reflected by the surface and the reflected ray has an angle α with the normal. The direction of the normal is an important characteristic since in the case of triangles, it determines which side reflects the light rays. If the normal of a triangle belonging to a surface is going to the outside of the object and the normal of another triangle of the same surface is going to the inside of the

object, then an observer outside the object will see some triangles reflecting light and some others reflecting nothing (because they are reflecting light inside the object).

So all normals should go outside the objects. It is the reason why all triangles should be defined in the same way in a file (clockwise or counter-clockwise) because the way a triangle is determined has an influence on the normal direction (see Figure 2-42). The first triangle is defined as *triangle (a, c, b)*, i.e. in clockwise order, and the other is defined as *triangle (a, b, c)*, i.e. in counter-clockwise order. In our program, the user has the opportunity to change the convention (clockwise or counter-clockwise) depending on the convention used in a particular file containing the surface description of objects so the program computes the normal for each triangle in the right direction [WRIGHT96], [WATT93].

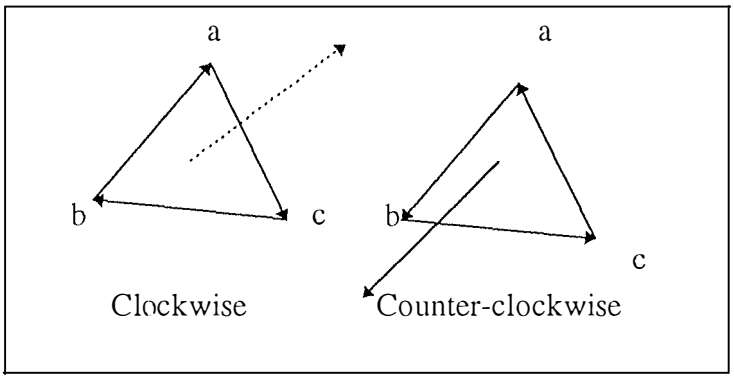


Figure 2-42 : Defining vertices order as clockwise and counter-clockwise

6.1.5 Views

The view of the scene is exactly what we see when we look at the window. It allows to see the scene between the near plane and the far plane. [SCHROEDER96] uses the term of **camera** when speaking about the view, since we can imagine a camera moving around an object and what we actually see is what we would see on a TV screen. Multiple views of the same object can be very useful, as lights and colors. Figure 2-43 shows the principle of a multiple view program. The program owns only one document (data) but is able to show it in different ways. A word processor for example would show in the first view the text in the way it would be printed, and another view that would be the text without any formatting, to increase the speed.

Indeed, the user can visualize the same object from the front side and the rear side in the same time. 3D Viewer doesn't limit the number of views, and every view can even receive different attributes, such as a different color or a different position in the 3D space (i.e. operations like rotation, translation and scaling can be applied to any view independently)

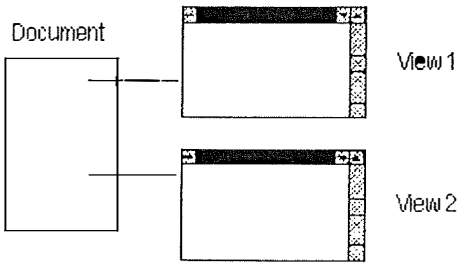


Figure 2-43 : Two views for the same document [MSDEV96]

Figure 2-44 shows the interface of 3D Viewer where the same object is displayed in two different windows with a different position in space and a different type of display (mesh and rendered surface).

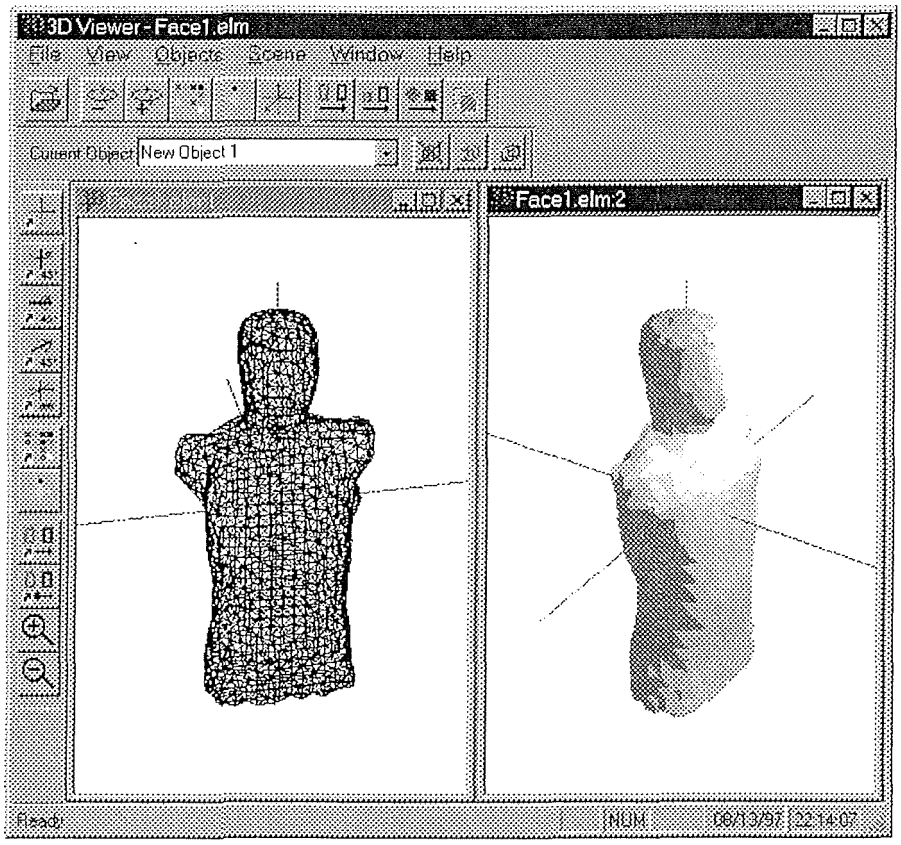


Figure 2-44 : 3D Viewer interface with multiple views

6.1.6 Cutting planes

Cutting plane is the last important feature we included in 3D Viewer. A cutting plane is a plane that cuts an object – or more often the whole scene – and removes everything that is above or

beneath, depending on settings. This feature allows to view inside an object that would be usually closed. We allow up to 6 different cutting planes, one for each side of the scene. Each of these planes can be manually set with a position in the scene, an angle, and different parameters allowing to show or not the plane, to cut or not the part of the scene.

Figure 2-45 shows the scene as shown in Figure 2-35 and three of the six cutting planes available. The top cutting plane can be lowered from its initial position to the initial position of the bottom cutting plane; the right cutting plane can be slid from its initial position to the initial position of the left cutting plane; the rear cutting plane can be pulled from its initial position to the initial position of the front cutting plane; and so on.

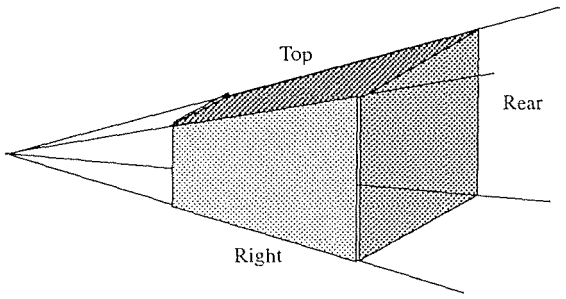


Figure 2-45 : Top, rear and right cutting planes

Figure 2-46 shows the same object (a small piece of bone of 2mm x 2mm x 2mm) displayed in 3D Viewer. The first image is the normal object, one cutting plane is showed in the second image and the plane cuts the half top of the third image.

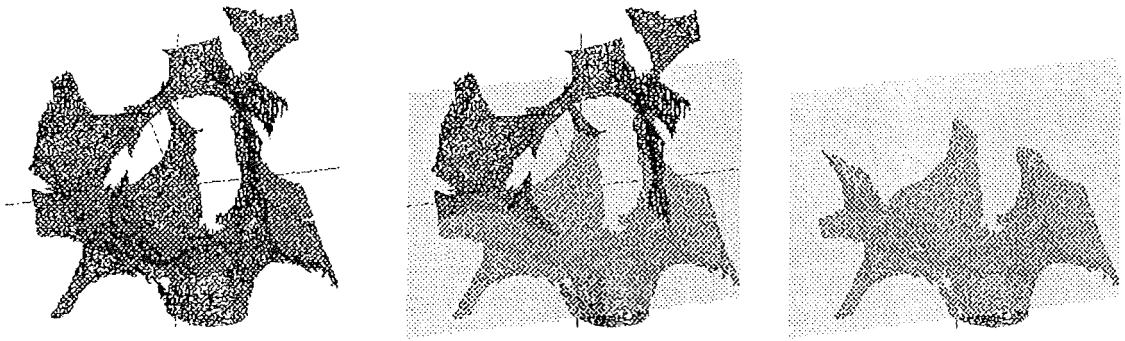


Figure 2-46 : Cutting planes

6.2 Program architecture.

As explained in [DUBOIS96], we will explain the architecture of the program, and especially the modularization which we think is enough to understand main parts of the program and how

they work together. First we will detail briefly main modules developed in 3D Viewer then we will show how they communicate and interact. Each module interface can be found in Appendix 4.

6.2.1 Main modules

The modularization has been done following two principles of cohesion :

- each module contains functions having same semantic or sharing same properties
- each module is object oriented

In future each module will be called an *object* because all of them actually are objects.

A. CScene

This is the most important object in the program. It contains[14] all the objects to be drawn, all parameters of the scene, such as lights, cuttings planes and far and near planes. This object is one of the biggest because the interface part of the object contains more than hundred functions allowing to change these parameters. Among these functions — also called methods — is the DrawScene[15] function which is called most of the time after each use of a dialog box allowing to change parameters. For example, when the dialog box for rotation is showed up and new parameters are entered, the scene has to be redrawn to take new changes in account.

B. CCuttingPlane

This object contains properties and methods for each of the six planes. Each plane is defined according to an equation which can be modified through a dialog box into the program. This object was one of the most difficult to write because of its complexity and poverty of documentation.

C. CLight

Like CCuttingPlane, this object contains properties and methods to describe each of the eight lights. As explained in the previous section, each light has ambiante, diffuse and specular properties and can be set at any spatial position and any color.

D. CReadMeshFile

The goal of this object is to read any kind of data, whatever it is a binary or an ASCII file, and transmits data read to the scene as a new object. Many functions appear the same even if they

[14] *contains* or *points to*

[15] This function corresponds to the semantic function "Display(Scene)" in the second part of this text.

are not and this object could have been developed using inheritance to reduce the code and to increase reusability.

E. CConverter

This object reads and writes binary and ASCII files to keep compatibility betwwen different applications such as 3D Viewer and 3dmesh for example.

F. VRMLMaker

This object saves the scene and creates a VRML[16] file which can be read by any web browser if the Live 3D plug-in is installed. We think that data interchange in the future will increase thanks to the web and an easy way to help this interchange is to allow to save in a worldwide recognized format such as VRML. Figure 2-47 shows the 3D viewer interface and the Netscape interface with the same object exported as VRML.

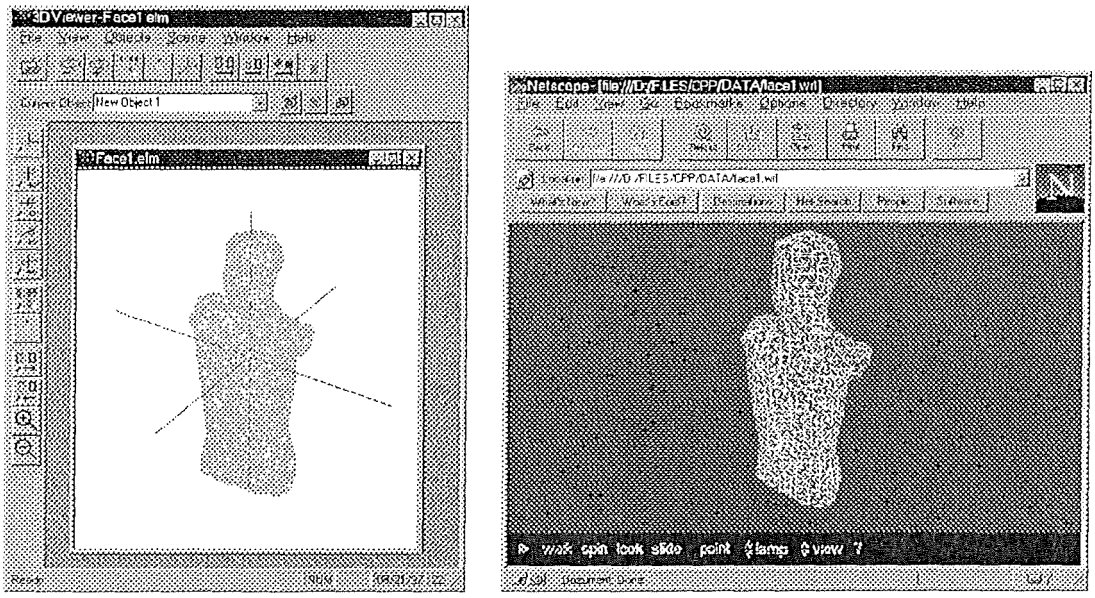


Figure 2-47 : 3D Viewer and an exported file in Netscape®[17]

[16] Virtual Reality Modeling Language, a language that allows to create 3D web sites, by Mark Pesce and Tony Parisi in 1994 [WRIGHT96]. See Appendix 5 for more details.

[17] Netscape, (Netscape Communications Corporation), a world-wide web brower, including Live3D plugin.

G. CDisplayBinary

This object loads a 1-bit binary image, displays it, allows the user to select a smaller region and saves it in the right format for a VTK process. This object is mainly based on a program written by Tony Keller is BASIC and was adapted in order to work under Windows.

H. IHM modules

These objects linking the scene with the new parameters are divided according to the rules explain in the TRIDENT methodology application. Indeed, there are about 15 dialog boxes, each of them corresponding to an object since it is the way Visual C++ works when creating a new dialog box.

I. CGLWorkApp

This module corresponds to the Windows application.

J. CGLWorkDoc

This object contains all methods and properties that correspond to the document. It is part of the MFC (Microsoft Foundation Class) and is defined as a class providing the basic functionality for user-defined document classes. A document represents the unit of data that the user typically opens with the File Open command and saves with the File Save command [MSDEV96]. In our case, the document will use the scene — typically, it owns a pointer to the scene — and allows to add and remove objects, counts the number of views available for the same scene.

K. CGLWorkView

[MSDEV96] says the CView[18] class provides the basic functionality for user-defined view classes. A view is attached to a document and acts as an intermediary between the document and the user: the view renders an image of the document on the screen or printer and interprets user input as operations upon the document. This object contains parameters proper to each view, for example, its color, its spatial position, default values for dialog boxes, such as rotation increase value, and the state of the view (rotating of not). Each CGLWorkView has a method to access the document in order to change properties of the scene. The methods listed in Appendix 4 correspond to the reaction to a menu. Their names are self-explaining and correspond to the concatenation of each word in the menu.

[18] CView is the base class, and CGLWorkView inherits of its properties. CView is part of MFC.

L. *CMainFrame*

According to [MSDEV96], the CMDIFrameWnd[19] class provides the functionality of a Windows Multiple Document Interface (MDI[20]) frame window, along with attributes for managing the window. The CMainFrame object correspond to the main window. Each window corresponding to a view will be a CMainFrame object.

M. *CIntList* and *CObjectList*

These objects handle lists, which are data structure. Methods allow adding, removing and finding objects in the lists.

6.2.2 Dependencies between modules

According to [DUBOIS96], the previous modules are to be classified into 5 classes. At the top level of the hierarchy we find the functionality modules. At level 4 are usually found HCI[21] and printing modules and level 3 contains data and data structures. Level 2 is dedicated to middle-ware modules such as DBMS[22] or Client/Server modules. Bottom level contains operating system modules. However, because the program is highly HCI and graphics oriented, and because the development is partially based on the TRIDENT methodology, we inverted level 5 and 4. HCI level is now top-level and functionality modules are are level 4. Description of dependencies is shown in Figure 2-48.

[19] CMDIFrameWnd is the base class for CMainFrame.

[20] A Multiple Document Interface allows more than one *document* to be open in the same time. MDI is different from a multiple view documents which allows multiple *windows* to be open in the same time but representing the same *document.*

[21] Human Computer Interface

[22] Databas Management Systems

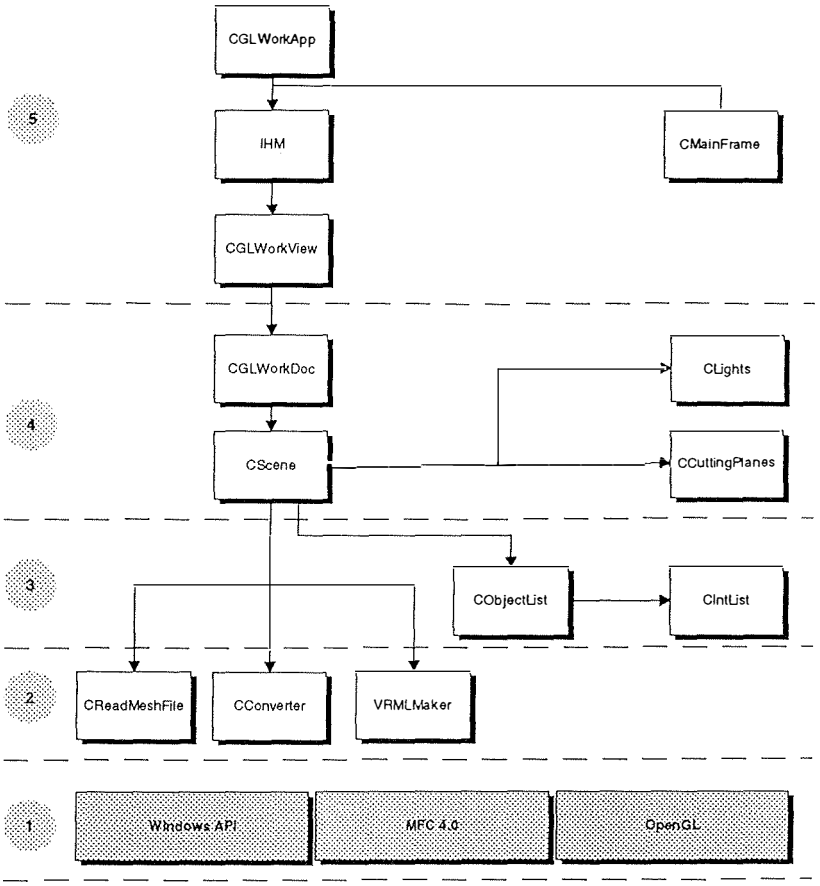


Figure 2-48 : Description of modules dependencies

7. Conclusion

This section is intented to summarize the whole process of visualization in the way we were asked to implement it. Figure 2-49 is showing the whole process in detail. Each grayed rectangle on the figure below corresponds to a section in this chapter. It is important to note that the rectangles outside of the grayed regions are not included in the visualization process but depend a lot on the results of the latter. With regard to the numerical analysis, it is in fact a strength analysis with a specialized software which take as input mesh files created by 3D Surface Maker or 3dmesh.

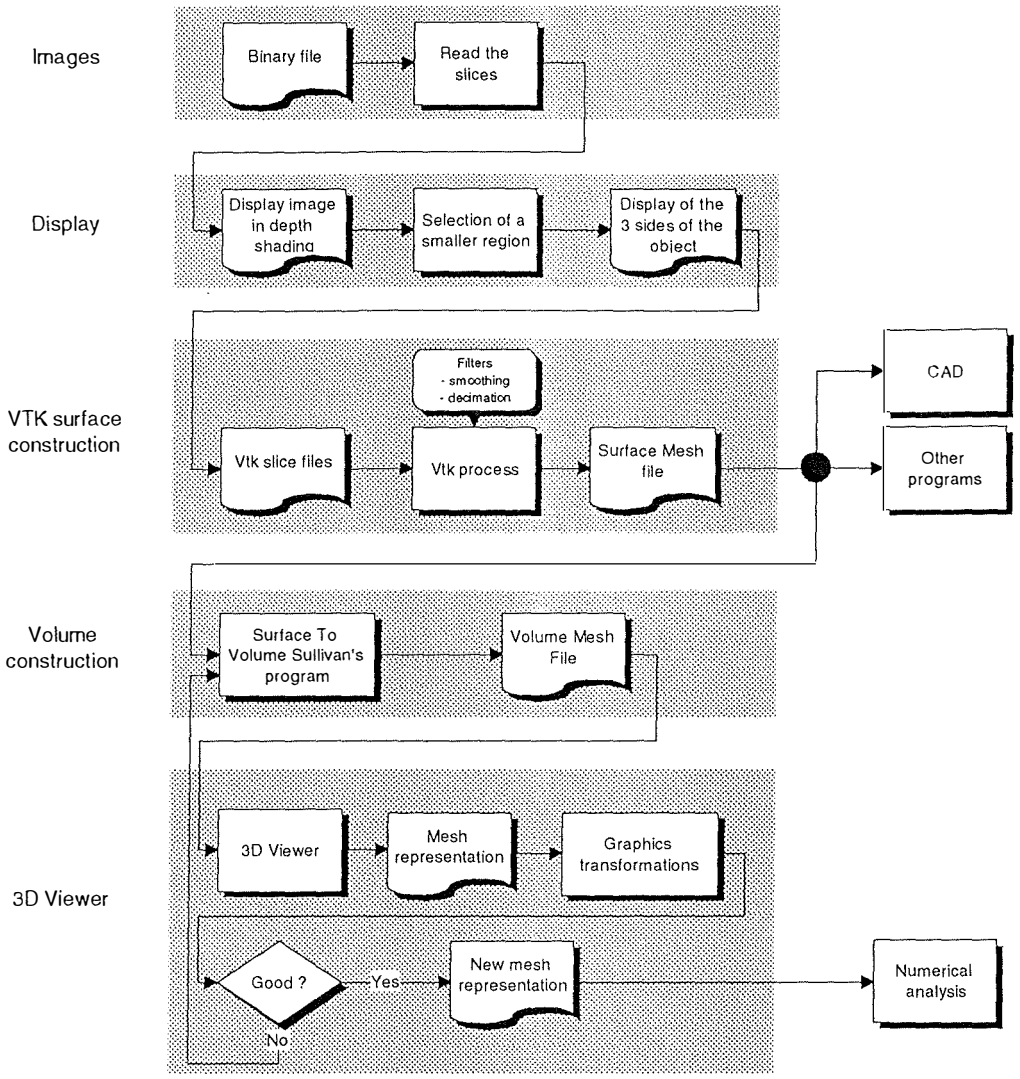


Figure 2-49 : The whole visualization process

Chapter 3 : 3D graphics technical considerations

1. OpenGL

The goal of this section is not to detail OpenGL neither to give a course on how to use it, but simply give main advantages of an easy to use, portable and powerful graphics library. OpenGL and VTK are two different graphics libraries, where VTK is one level higher. Indeed, the lowest layer of VTK is OpenGL, and this is a reason for which VTK is portable in many cases.

1.1 What is OpenGL ?

[WRIGHT96] defines OpenGL as *a software interface to graphics hardware, a 3D graphics and modeling library that is extremely portable and very fast.* OpenGL is relatively new on the market but seems to gain more and more following [WRIGHT96]. The advantage is that is was launched by the big graphics company in the world, Silicon Graphics, Inc. (SGI). These computers have more equipment than any other PC, especially optimized hardware for display of graphics. This hardware includes ultra-fast matrix transformations (see above, Matrix transformations). The word "Open" in OpenGL means that the library is open to the other computers, allowing easy adaptability to other platforms or operating systems. Indeed, the new release of Windows NT (NT 4.0) is including OpenGL, which means that any OpenGL-based program can run faster under Windows NT, especially if the computer owns a OpenGL compatible graphic card[23].

Features of OpenGL are enormous and it is not possible to even list them here. However, we will just point out most important ones. Of course OpenGL supports basics of 3D graphics,

[23] See "Graphics hardware" section for more details.

like lights, colors, cutting planes and matrix transformations but allows texture mapping[24], working with predefined complex objects like spheres or cylinders, giving visual effects like fog.

1.2 Portability

Portability for a powerful library is very important. A pure OpenGL program written in C++ under UNIX can be ported on a PC and run immediately without any change. The link between the code and the graphic card is made through the operating system. Under Windows NT and Windows 95, this link is made through two important dynamic libraries, OPENGL32.DLL and GLU32.DLL which are required to run any program based on OpenGL. The disadvantage, in our eyes, in that OpenGL does not include any high level functions that manage dialog boxes which makes quite impossible to develop a fully portable application.

2. The graphics hardware

2.1 Overview

Conditions to deal with graphics (not only 3D graphics) is to have a powerful computer because of underlying computations, a good graphic card (supporting one or more 3D standard) and a monitor. We are not going to talk about power of computers which would be out of the scope of this text, but we think that basics of how computer data are displayed on a monitor screen can be interesting.

2.2 Rasterization

[WRIGHT96] defines rasterization as the process of converting projected primitives and bitmaps into pixels. Figure 3-1 shows the result of rasterization for two lines. These lines are made of 2 points at their ends but to be displayed, these lines have to be converted into pixels. The grid shown in Figure 3-1 corresponds to a piece of the screen, where each square is a pixel. The grayed rectangles are what will be actually showed on the screen. This process is done anytime when a user draws a line with a drawing program such as Paint Brush or CorelDraw.

[24] Texture mapping is fitting pictures, images on a 3D object. A cube can be covered with a bitmap (an 2D image) to give a more realistic aspect. A example of texture mapping is computer games like DOOM where walls are covered with textures giving desired atmosphere. [WRIGHT96]

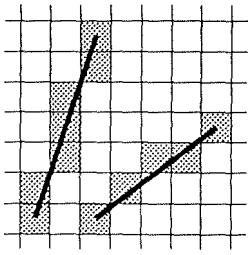


Figure 3-1 : Rasterization for 2 lines [WATT93]

Another characteristic of raster devices like computer screens or laser printers is their resolution [SCHROEDER96]. A laser printer for example is able to draw up to 600 pixels on one inch length (600 dpi) where an old matrix printer barely reach 50 dpi. A computer screen usually has a resolution of 80 pixels per inch, which allows about 1000 pixels wide by 800 pixels high.

2.3 Graphic boards

The graphic board is the interface between a computer and a monitor, and most of them now include 3D graphics acceleration, special chips to perform computation in place of the main processor [BYTE0896]. Graphic boards are linked with monitors and they have to share same capabilities in order to work at best. Owning a very good and power graphic board with a poor quality monitor is no use, and the opposite is also true.

These graphics boards are always equipped with memory which must be at least equal to the memory necessary to display the all screen. For example, with a resolution of 800x600 pixels, where each pixel is 16-bit encoded, 960000 bytes are required, that is, a one-megabyte memory graphic board is enough.

PART II

A critical application of the TRIDENT methodological framework.

During the design and development process of our program 3D Viewer (a prototype of a 3D visualization program), we did not use the TRIDENT methodology learned during our studies. The aim of this chapter is to carry out a critical application of the TRIDENT methodological framework described in [BODART95a] for the development of the 3D Viewer software human-computer interface and software architecture. We would like to find out where it presents weaknesses for the development of 3D medical imaging applications or applications with the same characteristics since it is primarily destined to design business oriented software. We point out that, to simplify the analysis, we will not talk about the possibility of different views of a scene that was implemented in the program.

The TRIDENT methodological framework is presented along 5 dimensions : (1) forming the user interface specifications from the output of the task analysis, (2) guiding the presentation design from ergonomic rules, (3) deriving the software architecture from the task analysis and the presentation components, (4) forming high level dialogue specifications from the output of the task analysis and (5) reducing the methodological framework to a specification framework. We will analyze the first three dimensions in the next chapters.

Chapter 4
First Dimension :
Graphical User Interface Specifications

1. Introduction

The task analysis is the very first step to the methodology. Before beginning with the analysis itself we have to correctly define the task. It consists in visualizing objects in three dimensions with different shapes, different positions, different colors... *with the intention of* helping the users to pose a diagnosis on these objects that are, in our case, pieces of trabecular bones. So it is a decision support task that comes within the scope of a bigger task that consists of a strength analysis of the elements previously listed. For more precision about this subject, see *Chapter 2 : Description of the process for the visualization program.* It is important not to confuse this task with the one that simply consists in displaying objects and where the only aim is the visualization itself. Our program could also be used for this purpose.

Since we did not observe the task and because we know nothing about the mental process of posing a medical diagnosis, we must presume the task. That is what we call a "**prescribed task**". The consequence of this fact, is that we can not analyze it. In view of the fact that the result of task analysis is useful for further steps in the TRIDENT methodology (derivation of the interaction styles , construction of the software architecture...) it is necessary to suppose some users' behaviors.

In [GOOSSENS95], the system implemented is presented as a toolbox where the user is able to use tools without specified order. We can also consider that the **toolbox metaphor** is suited to 3D Viewer because the task is weakly structured : it is prescribed and it is a decision making task. The same remark as in [GOOSSENS95] can be pointed out concerning the obvious necessity to use some tools before the others. For example, it is possible to manipulate an object if and only if it is previously loaded into a scene. Figure 4-1 is showing the tool box

metaphor where each tool correspond to functionality made available by the program and the house correspond the a scene (central element) on which the tools are applied. Figure 4-2 shows the same metaphor applied to Microsoft Word word processor (it actually works for all word processors !). The document is the central element and grammar, spelling, ... are the tools that can be applied to this document.

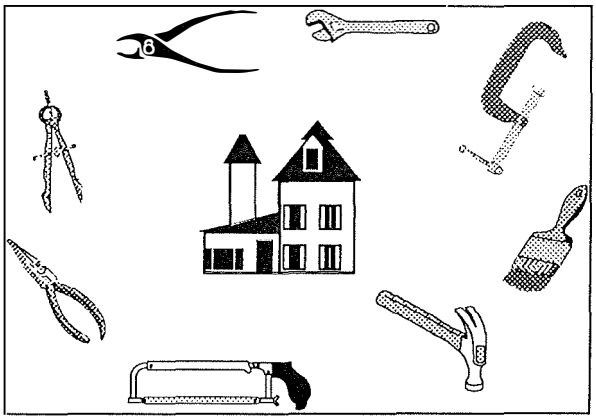


Figure 4-1 : The toolbox metaphor.

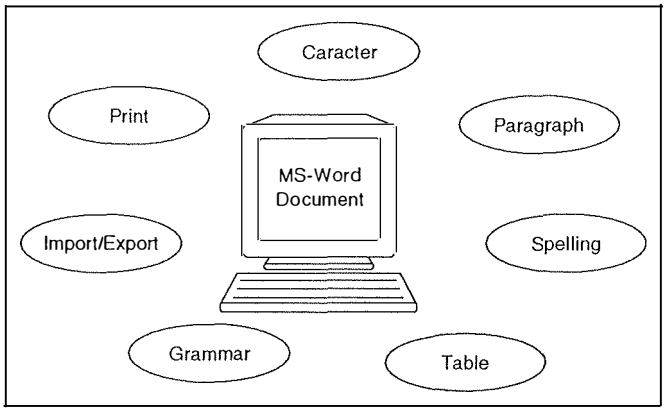


Figure 4-2 : Microsoft Word seen as a toolbox

2. Task "analysis"

The task "analysis" will be performed as follow : (1) the decomposition of the task into goals and sub-goals, (2) the decomposition into procedures, (3) the identification of the objects of the task, (4) the specification of the parameters relative to the task, (5) the description of the users stereotype and (6) the description of the environment where the task will be realized.

Task : to visualize 3D objects (pieces of bone...) with the intention of posing a diagnosis.

Role : scientific searcher.
Context : experiment.
Organization : University of Vermont, Musculoskeletal Research Lab.

2.1 Goals and sub-goals decomposition

The goals and sub-goals identified below are not coming from the observation of a task. They are rather the wishes expressed by our co-promoter in the United States, T. Keller.
Each goal and sub-goal has been labeled as this :

- (p) : preparation goal or sub-goal,
- (t) : transformation goal or sub-goal,
- (s) : selection goal or sub-goal.

The decomposition into goals and sub-goals is defined below :

1. Visualize objects with the intention of posing a diagnosis (t)
  1.1. Manage the scenes (t)
    1.1.1. Create a new scene (t)
      1.1.1.1. Create the scene (t)
      1.1.1.2. Make the new scene the current scene (p)
      1.1.1.1. Add an object into the new scene (t)
    1.1.2. Select the current scene (t)
    1.1.3. Remove the current scene (t)
    1.1.4. Specify parameters of the current scene (t)
      1.1.4.1. Specify the background color of the current scene (t)
      1.1.4.2. Specify the size of the current scene (t)
      1.1.4.3. Turn antialiasing[25] on/off (t)
      1.1.4.4. Turn culling face[26] on/off (t)
      1.1.4.5. Turn counterclockwise sorting on/off (t)
      1.1.4.6. Specify the shading method (t)

[25] Antialiasing : an algorithm to remove the distracting effects of point sampling a signal in the digital domain. (Real 3D, http://www.real3D.com)

[26] When culling face parameter is *on* the system does not compute hidden surfaces. This improves response speed.

1.1.5. Geometrically transform all the objects of the current scene (t)

1.1.5.1. Rotate along the longitude (t)
1.1.5.2. Translate horizontally (t)
1.1.5.3. Change the scale (t)

1.1.6. Cut a part of the current scene (t)

1.1.6.1. Choose the cutting planes (s)
1.1.6.2. Setting the cutting planes (p)
1.1.6.3. Show the cutting planes (p)
1.1.6.4. Cut the current scene (t)

1.1.7. Manage the lights (t)

1.1.7.1. Choose the lights (s)
1.1.7.2. Set the lights (p)
1.1.7.3. Turn the lights on/off (t)

1.1.8. Save the current scene in a VRML format (t)

1.2. Manage the objects of the current scene (t)

1.2.1. Add an object into the current scene (t)

1.2.1.1. Read the surface or the volume (t)
1.2.1.2. Give a name to the object (p)
1.2.1.3. Make the new object the current object (p)
1.2.1.4. Display the object in the current scene (p)

1.2.2. Select the current object (s)

1.2.3. Remove the current object from the current scene (t)

1.2.4. Change the name of the current object (t)

1.2.5. Get information about the current object (t)

1.2.5.1. Get the display type (s)
1.2.5.2. Get the number of tetrahedrons (s)
1.2.5.3. Get the number of triangles (s)
1.2.5.4. Get the number of vertices (s)
1.2.3.5. Get the file name (s)

1.2.6. Change the color of the current object (t)

1.2.6.1. Specify the object's color parameters (p)
1.2.6.2. Apply the color changing (t)

1.2.7. Change the type of visualization of the current object (t)

1.2.7.1. Choose a type of visualization (s)
1.2.7.2. Apply the type of visualization changing (t)

1.2.8. Show the current object axis (t)

1.2.9. Show the current object box (t)

The diagram of the goals and sub-goals decomposition is shown in Figure 4-3, Figure 4-4 and Figure 4-5.

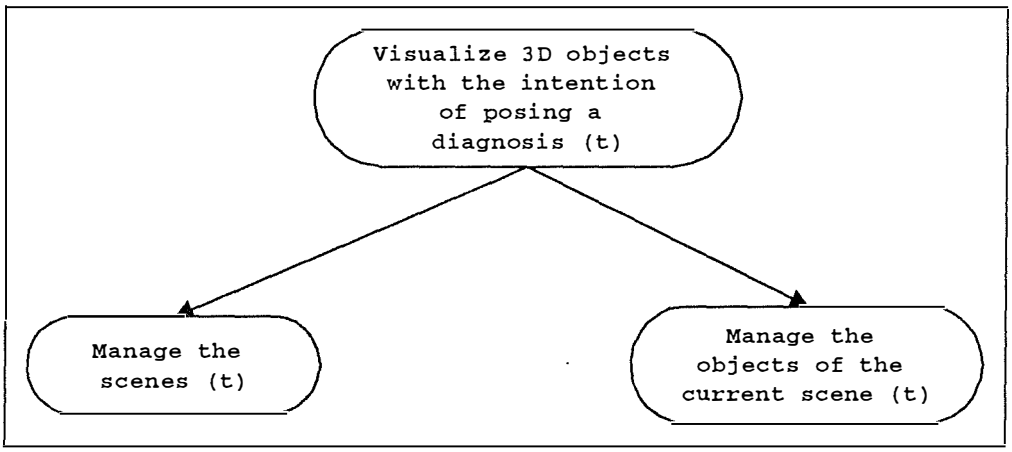


Figure 4-3 : The diagram of goal and sub-goal decomposition of the 3D visualization task

2.2 Procedures identification

The decomposition into goals and sub-goals shows us clearly that the main goal assigned to the task is composed of 2 sub-goals. Each of them is assigned to a sub-task :

- The management of the scenes
- The management of the objects

In this section, we are facing an inconvenience. The decomposition into procedures is aiming at showing how a process works. In our case, there is **no predetermined order** in the visualization process. As soon as the first object is loaded into the scene, any of the sub-tasks — for example cutting a part of the scene, changing the color of the object... or even adding a new object into the scene — can be carried out **at any time**. As already said, we call this the toolbox metaphor. They are two reasons why there is no process brought to the fore. First of all, the task that will be performed by the application is new and we did not observed it. We

need to analyze how the system is used by the users and, with the time, it will probably be possible to bring a certain process to the fore. One way to do this is to add hidden functions into the application aimed at writing into a file the sequence of actions executed by the user and then to carry out statistical analysis on this file. Secondly, by nature, the task is decision making oriented and this means that it is weakly structured.

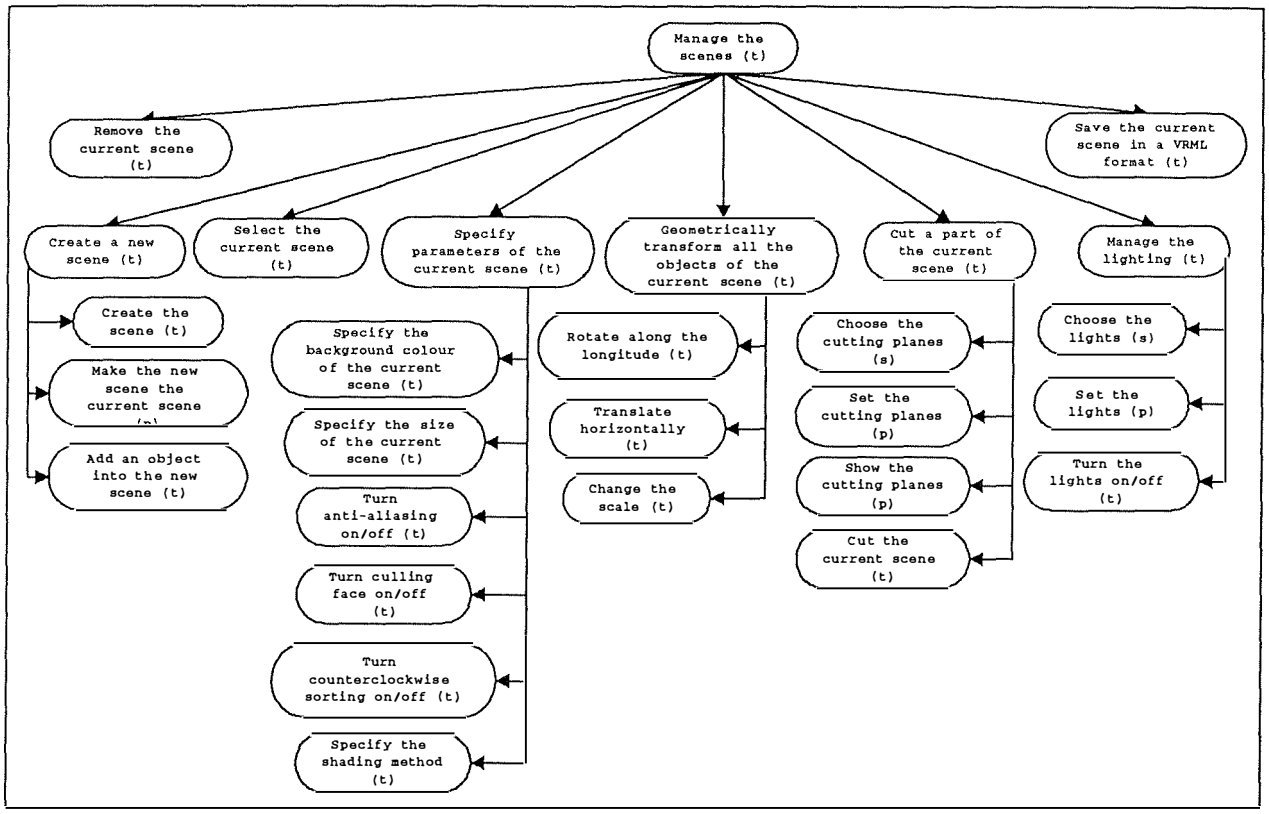


Figure 4-4 : The diagram of goal and sub-goal decomposition of the management of the scene sub-task

2.2.1 The management of the scenes

A. Creation of a new scene

current_scene ← Create_New_Scene ()
object_file_name ← Ask (user)
current_object ← Load_Object (object_file_name)
Add_Object (current_scene, current_object)

Select_Current_Object (current_scene, current_object)
new_name ← Ask (user)
Change_Name (current_scene, current_object, new_name)
Display (current_scene)

B. Selection of the current scene

current_scene ← Ask (user)
current_object ← Get_Current_Object (current_scene)

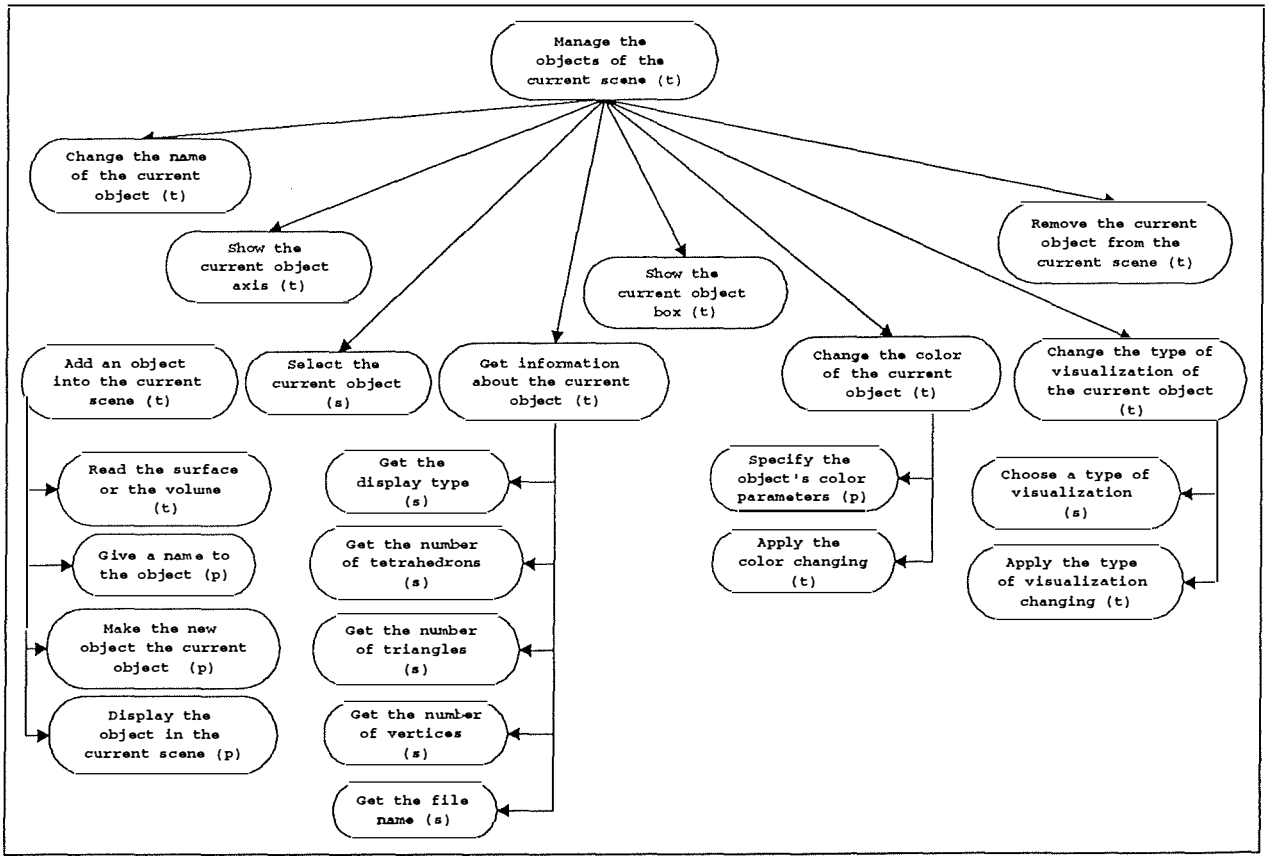


Figure 4-5 : The diagram of goal and sub-goal decomposition of the management of the objects sub-task

C. Removal of the current scene

new_current_scene ← Remove_Scene (current_scene)
current_scene ← new_current_scene
current_object ← Get_Current_Object (current_scene)

D. Specifying the parameters of the current scene

scene_color ← Ask (user)
Change_Background_Color (current_scene, scene_color)
near_plane ← Ask (user)
Change_Near_Plane (current_scene, near_plane)
view_angle ← Ask (user)
Change_View_Angle (current_scene, view_angle)
shading_method ← Ask (user)
Change_Shading_Method (current_scene, shading_method)
antialiasing ← Ask (user)
IF (antialiasing = **TRUE**)
THEN Turn_Antialiasing_On (current_scene)
ELSE Turn_Antialiasing_Off (current_scene)
END IF
culling_face ← Ask (user)
IF (culling_face = **TRUE**)
THEN Turn_Culling_Face_On (current_scene)
ELSE Turn_Culling_Face_Off (current_scene)
END IF
counter_clockwise_sorting ← Ask (user)
IF (counter_clockwise_sorting = **TRUE**)
THEN Sort_Counter_Clockwise (current_scene)
ELSE Sort_Clockwise (current_scene)
END IF
Display (current_scene)

E. Geometrical transformations of all the objects in the current scene

transformation_choice ← Ask (user)
// there are 3 transformations available : rotation, translation, scaling
IF (transformation_choice = "ROTATION")
THEN
  Get_Current_Rotation (current_scene, angle_x, angle_y, angle_z)
  rotation_x ← Ask (user)
  rotation_y ← Ask (user)
  rotation_z ← Ask (user)
  Rotate (current_scene, angle_x + rotation_x, angle_y + rotation_y, angle_z + rotation_z)
END IF

```
  IF (transformation_choice = "TRANSLATION")
  THEN
    translation_method ← Ask (user)
IF (translation_method = "BEST FIT")
THEN
    Get_Current_Translation (current_scene, "BEST FIT", pos_x, pos_y, pos_z)
    x_translation ← Ask (user)
    y_translation ← Ask (user)
    z_translation ← Ask (user)
    Translate (current_scene, "BEST FIT", pos_x + x_translation, pos_y + y_translation, pos_z
    + y_translation)
END IF
IF (translation_method = "ABSOLUTE")
THEN
    Get_Current_Translation (current_scene, "ABSOLUTE", pos_x, pos_y, pos_z)
    x_translation ← Ask (user)
    y_translation ← Ask (user)
    z_translation ← Ask (user)
    Translate (current_scene, "ABSOLUTE", pos_x + x_translation, pos_y + y_translation,
    pos_z + y_translation)
END IF
IF (translation_method = "RELATIVE")
THEN
    Get_Current_Translation (current_scene, "RELATIVE", pos_x, pos_y, pos_z)
    x_translation ← Ask (user)
    y_translation ← Ask (user)
    z_translation ← Ask (user)
    Translate (current_scene, "RELATIVE", pos_x + x_translation, pos_y + y_translation,
    pos_z + y_translation)
END IF
  END IF
  IF (transformation_choice = "SCALING")
  THEN
Get_Current_Scale (current_scene, cur_x_scale, cur_y_scale, cur_z_scale)
maintain_global_aspect_ratio ← Ask (user)
IF (maintain_global_aspect_ratio = TRUE)
THEN
```

x_y_z_scale ← Ask (user)
Change_Scale (scene, cur_x_scale * x_y_z_scale, cur_y_scale * x_y_z_scale, cur_z_scale * x_y_z_scale)
ELSE
x_scale ← Ask (user)
y_scale ← Ask (user)
z_scale ← Ask (user)
Change_Scale (current_scene, cur_x_scale * x_scale, cur_y_scale * y_scale, cur_z_scale * z_scale)
END IF
END IF
Display (current_scene)

F. *Cutting of a part of the current scene*

We will first notice that the sign "*" put behind a variable name means that several values can be put inside that variable. To illustrate, let's take the first variable "cutting_plane_choice*". At the first line, this variable is assigned one ore more values.

cutting_plane_choice* ← Ask (user)
// there are 6 cutting planes available : top, bottom, right, left, front or back plane
IF ("TOP" ∈ cutting_plane_choice*)
THEN
height_percentage ← Ask (user)
x_angle ← Ask (user)
z_angle ← Ask (user)
cutting_plane ← Define_Cp ("TOP", height_percentage, x_angle, z_angle)
show ← Ask (user)
IF (show = **TRUE**)
THEN
color ← Ask (user)
size ← Ask (user)
grid ← Ask (user)
IF (grid = **TRUE**)
THEN
number_of_wires ← Ask (user)
Display_Cp (current_scene, cutting_plane, color, size, "WIRES", number_of_wires)

```
        ELSE
            translucence_percentage ← Ask (user)
            Display_Cp (current_scene, cutting_plane, color, size, "TRANSLUCENCE",
            translucence_percentage)
        END IF
    END IF
    cut ← Ask (user)
    IF (cut = TRUE)
    THEN Cut_Scene (current_scene, cutting_plane)
    END IF
     END IF
     IF ("BOTTOM" ∈ cutting_plane_choice*)
     THEN
    height_percentage ← Ask (user)
    x_angle ← Ask (user)
    z_angle ← Ask (user)
    cutting_plane ← Define_Cp ("BOTTOM", height_percentage, x_angle, z_angle)
    show ← Ask (user)
    IF (show = TRUE)
    THEN
        color ← Ask (user)
        size ← Ask (user)
        grid ← Ask (user)
        IF (grid = TRUE)
        THEN
            number_of_wires ← Ask (user)
            Display_Cp (current_scene, cutting_plane, color, size, "WIRES", number_of_wires)
        ELSE
            translucence_percentage ← Ask (user)
            Display_Cp (current_scene, cutting_plane, color, size, "TRANSLUCENCE",
            translucence_percentage)
        END IF
    END IF
    cut ← Ask (user)
    IF (cut = TRUE)
    THEN Cut_Scene (current_scene, cutting_plane)
    END IF
```

```
  END IF
  IF ("RIGHT" ∈ cutting_plane_choice*)
  THEN
length_percentage ← Ask (user)
y_angle ← Ask (user)
z_angle ← Ask (user)
cutting_plane ← Define_Cp ("RIGHT", length_percentage, y_angle, z_angle)
show ← Ask (user)
IF (show = TRUE)
THEN
    color ← Ask (user)
    size ← Ask (user)
    grid ← Ask (user)
    IF (grid = TRUE)
    THEN
        number_of_wires ← Ask (user)
        Display_Cp (current_scene, cutting_plane, color, size, "WIRES", number_of_wires)
    ELSE
        translucence_percentage ← Ask (user)
        Display_Cp (current_scene, cutting_plane, color, size, "TRANSLUCENCE",
        translucence_percentage)
    END IF
END IF
cut ← Ask (user)
IF (cut = TRUE)
THEN Cut_Scene (current_scene, cutting_plane)
END IF
  END IF
  IF ("LEFT" ∈ cutting_plane_choice*)
  THEN
length_percentage ← Ask (user)
y_angle ← Ask (user)
z_angle ← Ask (user)
cutting_plane ← Define_Cp ("LEFT", length_percentage, y_angle, z_angle)
show ← Ask (user)
IF (show = TRUE)
THEN
```

color ← Ask (user)
size ← Ask (user)
grid ← Ask (user)
IF (grid = **TRUE**)
THEN
number_of_wires ← Ask (user)
Display_Cp (current_scene, cutting_plane, color, size, "WIRES", number_of_wires)
ELSE
translucence_percentage ← Ask (user)
Display_Cp (current_scene, cutting_plane, color, size, "TRANSLUCENCE", translucence_percentage)
END IF
END IF
cut ← Ask (user)
IF (cut = **TRUE**)
THEN Cut_Scene (current_scene, cutting_plane)
END IF
END IF
IF ("FRONT" ∈ cutting_plane_choice*)
THEN
depth_percentage ← Ask (user)
x_angle ← Ask (user)
y_angle ← Ask (user)
cutting_plane ← Define_Cp ("FRONT", depth_percentage, x_angle, y_angle)
show ← Ask (user)
IF (show = **TRUE**)
THEN
color ← Ask (user)
size ← Ask (user)
grid ← Ask (user)
IF (grid = **TRUE**)
THEN
number_of_wires ← Ask (user)
Display_Cp (current_scene, cutting_plane, color, size, "WIRES", number_of_wires)
ELSE
translucence_percentage ← Ask (user)

```
        Display_Cp (current_scene, cutting_plane, color, size, "TRANSLUCENCE",
        translucence_percentage)
    END IF
END IF
cut ← Ask (user)
IF (cut = TRUE)
THEN Cut_Scene (current_scene, cutting_plane)
END IF
  END IF
  IF ("BACK" ∈ cutting_plane_choice*)
  THEN
depth_percentage ← Ask (user)
x_angle ← Ask (user)
y_angle ← Ask (user)
cutting_plane ← Define_Cp ("BACK", depth_percentage, x_angle, y_angle)
show ← Ask (user)
IF (show = TRUE)
THEN
    color ← Ask (user)
    size ← Ask (user)
    grid ← Ask (user)
    IF (grid = TRUE)
    THEN
        number_of_wires ← Ask (user)
        Display_Cp (current_scene, cutting_plane, color, size, "WIRES", number_of_wires)
    ELSE
        translucence_percentage ← Ask (user)
        Display_Cp (current_scene, cutting_plane, color, size, "TRANSLUCENCE",
        translucence_percentage)
    END IF
END IF
cut ← Ask (user)
IF (cud = TRUE)
THEN Cut_Scene (current_scene, cutting_plane)
END IF
  END IF
  Display (current_scene)
```

G. Management of the lights

```
working_with_lights ← Ask (user)
IF (working_with_lights = TRUE)
THEN
FOR EACH light_i DO
    // They are maximum 8 lights available, each one identified by a number (light_i)
    on ← Ask (user)
    IF (on = TRUE)
    THEN
        color ← Ask (user)
        specular_component ← Ask (user)
        far ← Ask (user)
        IF (far = TRUE)
        THEN
            x_direction ← Ask (user)
            y_direction ← Ask (user)
            z_direction ← Ask (user)
            Turn_Light_On (current_scene, light_i, color, specular_component, "FAR",
            x_direction, y_direction, z_direction)
        ELSE
            x_position ← Ask (user)
            y_position ← Ask (user)
            z_position ← Ask (user)
            Turn_Light_On (current_scene, light_i, color, specular_component, "NEAR",
            x_position, y_position, z_position)
        END IF
    END IF
    ELSE Turn_Light_Off (current_scene, light_i)
END FOR
 ELSE
Turn_All_Lights_Off (current_scene)
 END IF
 Display (current_scene)
```

H. Saving into VRML format

```
vrml_file_name ← Ask (user)
Save_To_Vrml (current_scene, vrml_file_name)
```

2.2.2 The management of the objects

A. Addition of an object into the current scene

```
object_file_name ← Ask (user)
current_object ← Load_Object (object_file_name)
Add_Object (current_scene, current_object)
Select_Current_Object (current_scene, current_object)
new_name ← Ask (user)
Change_Name (current_scene, current_object, new_name)
Display (current_scene)
```

B. Selection of the current object

```
object_name ← Ask (user)
current_object ← Get_Object (current_scene, object_name)
Select_Current_Object (current_scene, current_object)
```

C. Removal of the current objet from the current scene

```
confirmation ← Ask (user)
IF (confirmation = TRUE)
THEN
new_current_object ← Remove_Object (current_scene, current_object)
current_object ← new_current_object
Select_Current_Object (current_scene, current_object)
Display (current_scene)
END IF
```

D. Changing the name of the current object

```
new_name ← Ask (user)
Change_Name (current_scene, current_object, new_name)
```

E. Getting information about an object

```
object_name ← Ask (user)
object ← Get_Object (current_scene, object_name)
number_of_vertices ← Get_Number_Of_Vertices (current_scene, object)
file_name ← Get_File_Name (current_scene, object)
visualization_type ← Get_Visualization_Type (current_scene, object)
IF (visualization_type = "TRIANGLES")
```

THEN number_of_elements ← Get_Number_Of_Triangles (current_scene, object)
END IF
IF (visualization_type = "TETRAHEDRONS")
THEN number_of_elements ← Get_Number_Of_Tetrahedrons (current_scene, object)
END IF
IF (visualization_type = "POINTS CLOUD")
THEN number_of_elements ← 0
END IF

F. Changing the color of the current object

color ← Ask(user)
specular_component ← Ask(user)
shininess ← Ask(user)
translucence ← Ask(user)
Change_Color (current_scene, current_object, color, specular_component, shininess, translucence)
Display (current_scene)

G. Changing the type of visualization of the current object

visualization_type ← Ask(user)
// There are 3 types of visualization : points cloud, triangles or tetrahedrons
Change_Visualization_Type (current_scene, current_object, visualization_type)
Display (current_scene)

H. Showing the current object axis

show ← Ask(user)
IF (show = **TRUE**)
THEN
color ← Ask(user)
length ← Ask(user)
move_with_object ← Ask(user)
Show_Axis_On (current_scene, current_object , color, length, move_with_object)
ELSE
Show_Axis_Off (current_scene, current_object)
END IF
Display (current_scene)

1. Showing the current object box

show ← Ask(user)
IF (show = **TRUE**)
THEN
color ← Ask(user)
size ← Ask(user)
Show_Box_On (current_scene, current_object, color, size)
ELSE
Show_Box_Off (current_scene, current_object)
END IF
Display (current_scene)

2.3 Identification of the objects of the task

From the decomposition into procedures, we notice that there are seven important objects. The **scene** contains **objects**, each of them being made of **points** defined in a three dimensional space represented by the scene. Each object has a name, some material properties — i.e. color properties : specular component of the color, shininess... [27]— and is stored in a file. There are several file formats but, in general, they all contain a set of points — the coordinates — called also vertices and a set of **connectivities** — lines between two points. The connectivities either represent **triangles** for a surface or **tetrahedrons** for a volume (Cf. *Chapter 2 : Description of the process for the visualization program*). The objects can be displayed as a surface mesh, as a volume mesh or as a points cloud.

The scene contains both **cutting planes** and **lights**. The cutting planes permit the user to see inside objects and help them to have a better idea of the shape of this object. There are six cutting planes corresponding to the six sides of the volume defined by the scene, each one with a name and a number of characteristics : the way they appear on screen — translucid or grid plane — , the transparency percentage in the case of a translucid plane or the number of wires for a grid plane, a color, the size — percentage of the size of one side of the scene. A cutting plane can be visible, i.e. that we can see the grid or translucid plane on the screen. It can also be active or not. When it is, you don't see the cut part on the screen. Each cutting plane is situated at a distance percentage from the side of the scene it is belonging to and has two angles with this side. If both angles are equal to zero, the cutting plane is parallel to the side.

[27] See Lights in *Chapter 2 : Description of the process for the visualization program*

The lights components[28] interact with the objects color components[29] and give more reality to the scene. With good lights and objects material parameters setting you can better evaluate the shape of the objects. There are up to eight lights allowed in the scene. One light can be far in which case its direction has to be specified. When it is near, the light position in the scene has to be specified. A light can be turned on or off and is defined with a color and the specular component of the color.

The scene has some characteristics : the background color and an angle of view in the vertical direction. Anti-aliasing and culling face can be performed or not. (See note 25 p.85 and note 26 p.85)It's possible to decide that, in the case of objects made up of triangles, the points that compose these triangles are sorted in a clockwise or in a counter-clockwise way. It is possible to carry out geometrical transformations — scaling, translation and rotation — on the scene. The transformations apply to all the objects that are contained in the scene. So the scene is also characterized by the scaling state, the translation state and the rotation state.

2.4 Parameters relative to the task

We will specify the parameters relative to the task as explained in detail in [VANDERDONCKT93a]. Seven parameters will be "analyzed" : (1) the prerequisite needed to perform the task, (2) the productivity of the task, (3) the existence of an objective environment, (4) the practicability of the objective environment reproducibility, (5) the structuration of the task, (6) the importance of the task and (7) the complexity of the task.

The "3D visualization with the intention of posing a diagnosis" task is performed before the numerical analysis task and plays a role as a support to the decision process. As already said, it is a task we did not observed. As a consequence, we cannot analyze the seven parameters. All that we can do, it is just **presume** them.

2.4.1 Prerequisite

The user needs only some basic knowledge of the Microsoft Windows 95 or NT environment and the ability to manipulate a mouse. About the concepts used in the program such as the color properties of the object, lighting properties of the lights... we think that the learning period should be short since the result of the application of these concepts is directly visible on the screen. As a consequence, it is normal to presume that the prerequisite is **low**.

---

[28] See Lights in *Chapter 2 : Description of the process for the visualization program*

[29] See The (objects) color in *Chapter 2 : Description of the process for the visualization program*

2.4.2 Productivity

We presume that the productivity should be **moderate** because the goal of the program is that the users do not waste time with manipulations that are irrelevant to the task fulfillment and, on the other hand, there is no performance constraint. Indeed, imagine that the user wants to have an idea about the inside of a piece of trabecular bone. He would like to see if it is full or if there are many holes. In this case, he will manipulate at least a cutting plane, configure it as easily as possible — its relative position from one side of the scene, its slope with this side of the scene... If he needs to cut the piece of bone at an accurate position, he can display the cutting plane to help him to position the cutting plane. When the cutting plane is not well placed, the user can configure it again from the last position.

2.4.3 Objective environment

The environment exists under a form that the user can directly manipulate. He can observe a piece of bone directly with his eyes, surely through a microscope, he can cut them to see inside, he can adjust lights to help him to better estimate the shape of the piece... Therefore we think that the objective environment is **existent**.

2.4.4 Environment reproductibility

The environment reproducibility is **practicable.** The objective environment exists and can be reproduced in our application. The scene displays, in our case, pieces of bone because the mental decision process is based on the visual aspect of these objects. The notion of light is present. We can also cut a part of a scene and as a consequence, cut the piece of bone that is inside the scene.

2.4.5 Task structuration

We can not tell how the mental process of posing a diagnosis on a a piece of bone is structured. Since it is a decision making task, we think it is not well structured. In view of the fact that the visualization of the objects is aiming at helping the user to pose a diagnosis on these objects, we presume that the structuration of this task should be **low**. Furthermore, we did not observed the task, so how could we impose a structuration on it ? It is the reason why we decided to use the toolbox model (see Figure 4-1).

2.4.6 Task importance

Again, we can't tell the importance of the task. Since we have been asked to write an application to help to perform the task, we suppose that its importance should be **high**.

2.4.7 Task complexity

As regards the manipulation we did not want the task to be complex. Only the manipulation of the mouse and the manipulation of the keyboard are necessary to use the program. In the point of view of the intellectual complexity, it is impossible for us to give any idea. Is posing a diagnosis on the quality of a piece of bone easy or not ? As a consequence, we do not take risks and we presume that the task complexity is **moderate**.

2.5 Users stereotype description

Just as the specification of the parameters relative to the task, the description of the users stereotype will be performed as explained in detail in [VANDERDONCKT93a]. Four parameters will be inspected : (1) the users' experience in carrying out the task, (2) the users' experience in using information systems, (3) the users' motivation and finally (4) the users' experience in the use of complex interaction means.

We recognize only one sort of users. They are researchers in biomedical engineering and are experts on the use of computerized systems more complicated than the 3D Viewer. Even if the 3D visualization task with the intention of posing a diagnosis is new in the scientific experiment process, these users will learn easily and rapidly to use the system that will help to carry out the task.

2.5.1 Experience of the task

We can not tell how many times the users already analyzed pieces of trabecular bones without the use of a 3D visualization software. On the other hand, we think that they never carried out the task with the help of a 3D visualization program. It is the reason why we suppose that their experience of the task is **elementary.**

2.5.2 Experience of systems

Their experience of information systems is **rich**. The users' experience level of the use of an information system is the one of an expert.

2.5.3 Motivation

We suppose that their motivation is **high**. The task belongs to a scientific experiment process whose results are interesting for them.

2.5.4 Experience of complex interaction means

The users' experience of the use of complex interaction means is considered as **rich**. They have at least a great ability in using keyboards and mice. They are also able to use scientific scanners, cameras and other interaction means.

2.6 Environment description

We will describe the environment or workplace still in function of what is explained in [VANDERDONCKT93a]. We are going to consider two parameters : (1) the type of processing and (2) the capacity of processing.

2.6.1 Processing type

We have been asked to implement the software so that it works under Windows 95 and NT operating system which are multi-processing environments. While they are working with the visualization process, users can also carry out other tasks such as volume or surface creation. They can also work on numerical analysis or can do whatever task they want to perform. In conclusion the type of treatment is **multi-processing**.

2.6.2 Processing capacity

We think that the treatment capacity is **moderate** to **high** except for enormous files where it can take seconds to realize any operation and where most of the CPU resources are used.

3. Expressing the product of the task analysis

The task analysis leads to four next steps : (1) the construction of the entity-relationship model, (2) the identification of the the semantic functions of the application, (3) the composition of the

activity chaining graph and (4) the derivation of the dialogue attributes with the interaction styles.

3.1 Entity-relationship model

The entity-relationship model is explained in detail in [BODART95a]. The entities are represented by rectangles with the entity name on top of them and the list of entity attributes following the name. When an attribute is identifying an entity, it is underlined. The relations between entities are represented by hexagons with the relation name on top of them and can have attributes. Each entity linked to another one by a relation plays a role which has a name and cardinalities. On this model we added the notion of ISA-relation, represented by a triangle connecting an entity to a specialization of it. The cutting plane, for example, which is represented by en entity entitled "CUTTING_PLANE" is specialized in "FRONT_BACK_CP", entity that represents the front or back cutting planes. The "FRONT_BACK_CP" entity inherits the attributes of the "CUTTING_PLANE" entity. Figure 4-6 shows the diagram. We are now going to specify the integrity constraints.

Constraint 1 : *CP_Transparency* of *SHOWING_CP* has a value if and only if *CP_Grid* of *CUTTING_PLANE* is false.

Constraint 2 : *CP_Wires* of *SHOWING_CP* has a value if and only if *CP_Grid* of *CUTTING_PLANE* is true.

Constraint 3 : *LT_Coordinates* of *LIGHT* represent the direction of the rays of the light if and only if *LT_Far* of *LIGHT* is true else they represent the position of the same light.

Constraint 4 : If *CP_Name* of *CUTTING_PLANE* = "BACK" or "FRONT" then *CUTTING_PLANE* is a *FRONT_BACK_CP*

Constraint 5 : If *CP_Name* of *CUTTING_PLANE* = "LEFT" or "RIGHT" then *CUTTING_PLANE* is a *LEFT_RIGHT_CP*

Constraint 6 : if *CP_Name* of *CUTTING_PLANE* = "TOP" or "BOTTOM" then *CUTTING_PLANE* is a *TOP_BOTTOM_CP*

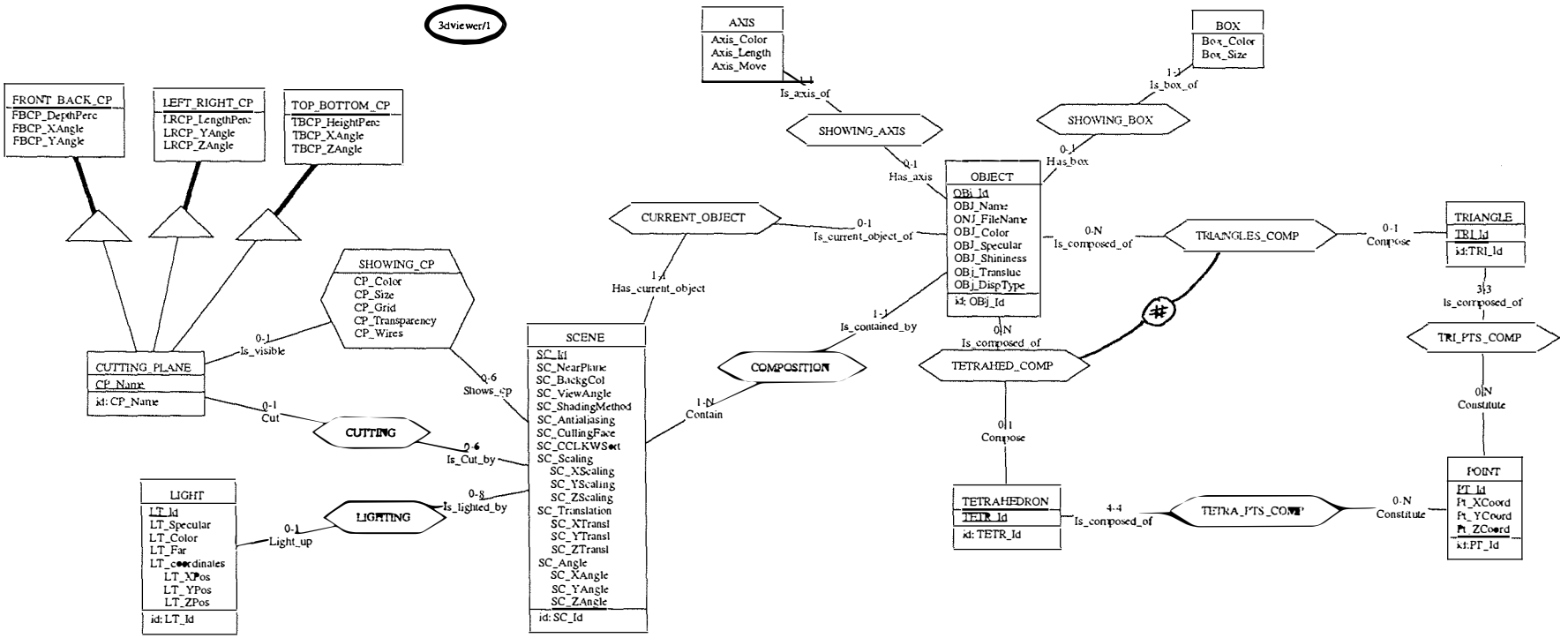


Figure 4-6 : The ERA model

3.2 *Identification of the semantic functions*

The semantic functions are directly coming from the procedures identified in *Section 2.2*. That is to say there is no abstraction mechanism (actions ≡ functions) because we do not know the task we did not observed and, carrying out the task analysis *a posteriori*, we tend to think of the actions in term of computer functions. We will list the procedures with the functions they are using and give a description of these functions, among others their relation with the ERA model (Figure 4-6).

3.2.1 Creation of a new scene

- *Create_New_Scene ()*

  Goal : Create a new scene and give it default attributes.
  Input : /
  Output : The scene newly created.
  Description : The function creates an occurrence of the *SCENE* entity, updates its *SC_Id* attribute with the identifier of the scene and updates all the other attributes with default values.

- *Load_Object (file_name)*

  Goal : Create a new object and give it default attributes.
  Input : The name of the connectivities and the nodes files.
  Output : The object newly created.
  Description : The function creates an occurrence of the *OBJECT* entity, updates its *OBj_Id* attribute with the identifier of the object, updates its *OBj_FileName* attribute with *object_file_name* and updates all the other attributes with default values. It reads the connectivities (*file_name*.elm) file and the nodes file (*file_name*.nod), and constructs the objects. The last action means that it creates as many occurrences of the *POINT* entity as there are nodes in the *object_file_name*.nod file, updates each attribute with the values contained in the file.
  If the files contain the definition of the surface mesh of the object it creates as many occurrences of the *TRIANGLE* entity as there are triangles defined in the *file_name*.elm file, updates their attribute, constructs the necessary *TRI_PTS_COMP* and *TRIANGLES_COMP* relations.
  If the files contain the definition of the volume mesh of the object it creates as many occurrences of the *TETRAHEDRONS* entity as there are tetrahedrons

defined in the *file_name*.elm file, updates their attribute, constructs the necessary *TETRA_PTS_COMP* and *TETRAHED_COMP* relations.

- *Add_Object (scene, object)*

  Goal : Add an object into a scene.
  Input : The scene and the object.
  Output : The scene modified.
  Description : The function creates a new occurrence of the *COMPOSITION* relation between the *scene* occurrence of the *SCENE* entity and the *object* occurrence of the *OBJECT* entity.

- *Select_Current_Object (scene, object)*

  Goal : Select an object as being the current one in the scene.
  Input : The scene and the object.
  Output : The scene modified.
  Description : The function eventually destroys the only *CURRENT_OBJECT* relation and creates a new occurrence of the *CURRENT_OBJECT* relation between the *scene* occurrence of the *SCENE* entity and the *object* occurrence of the *OBJECT* entity.

- *Change_Name (scene, object, name)*

  Goal : Change the name of an object contained in a scene.
  Input : The scene, the object and the new name.
  Output : The scene modified.
  Description : The function updates the *OBj_Name* attribute of the *object* occurrence of the *OBJECT* entity, which *Is_contained_by* the *scene* occurrence of the *SCENE* entity, with *name*.

- *Display (scene)*

  Goal : Display the scene into a window on the screen.
  Input : The scene (a logical description).
  Output : /
  Description : The function interprets the ERA model into drawing primitives and executes these primitives.

3.2.2 Selection of the current scene

- *Get_Current_Object (scene)*

  Goal : Get the current object contained in a scene.

Input : The scene.
Output : The current object contained in the scene.
Description : The function returns the occurrence of the *OBJECT* entity that *is_current_object_of* the *scene* occurrence of the *SCENE* entity.

3.2.3 Removal of the current scene

- *Remove_Scene (scene)*
  Goal : Remove a scene.
  Input : The scene.
  Output : Another scene.
  Description : The function destroys all the occurrences of all the entities and all the relations that are in connection with the *scene* occurrence of the *SCENE* entity and returns another occurrence of the *SCENE* entity.

- *Get_Current_Object (scene)*
  See item "2. Selection of the current scene" of this section.

3.2.4 Specifying the parameters of the current scene

- *Change_Background_Color (scene, color)*
  Goal : Change the background color of a scene.
  Input : The scene and the background color value.
  Output : The scene modified.
  Description : The function updates the *SC_BackgCol* attribute of the *scene* occurrence of the *SCENE* entity with the *color*.

- *Change_Near_Plane (scene, near_plane)*
  Goal : Change the near plane value of a scene.
  Input : The scene and the near plane value (between 1 and 20).
  Output : The scene modified.
  Description : The function updates the *SC_NearPlane* attribute of the *scene* occurrence of the *SCENE* entity with the *near_plane*.

- *Change_View_Angle (scene, view_angle)*
  Goal : Change the view angle value of a scene.
  Input : The scene and the view angle value (between 10 and 120).
  Output : The scene modified.

Description : The function updates the *SC_ViewAngle* attribute of the *scene* occurrence of the *SCENE* entity with the *view_angle.*

- *Change_Shading_Method (scene, shading_method)*

  Goal : Change the shading method used in a scene.
  Input : The scene and the shading method (flat or smooth).
  Output : The scene modified.
  Description : The function updates the *SC_ShadingMethod* attribute of the *scene* occurrence of the *SCENE* entity with the *shading_method.*

- *Turn_Antialiasing_On (scene)*

  Goal : Turn the anti-aliasing method on in a scene.
  Input : The scene.
  Output : The scene modified.
  Description : The function updates the *SC_AntiAliasing* attribute of the *scene* occurrence of the *SCENE* entity with the value TRUE.

- *Turn_Antialiasing_Off (scene)*

  Goal : Turn the anti-aliasing method off in a scene.
  Input : The scene.
  Output : The scene modified.
  Description : The function updates the *SC_AntiAliasing* attribute of the *scene* occurrence of the *SCENE* entity with the value FALSE.

- *Turn_Culling_Face_On (scene)*

  Goal : Turn the culling face method on in a scene.
  Input : The scene.
  Output : The scene modified.
  Description : The function updates the *SC_CullingFace* attribute of the *scene* occurrence of the *SCENE* entity with the value TRUE.

- *Turn_Culling_Face_Off (scene)*

  Goal : Turn the culling face method off in a scene.
  Input : The scene.
  Output : The scene modified.
  Description : The function updates the *SC_CullingFace* attribute of the *scene* occurrence of the *SCENE* entity with the value FALSE.

- *Sort_Counter_Clockwise (scene)*

  Goal : Turn the counter clockwise sorting method on in a scene.
  Input : The scene.
  Output : The scene modified.
  Description : The function updates the *SC_CCLKWSort* attribute of the *scene* occurrence of the *SCENE* entity with the value TRUE.

- *Sort_Clockwise (scene)*

  Goal : Turn the counter clockwise sorting method on in a scene.
  Input : The scene.
  Output : The scene modified.
  Description : The function updates the *SC_CCLKWSort* attribute of the *scene* occurrence of the *SCENE* entity with the value FALSE.

- *Display (scene)*

  See item "1. Creation of a new scene" of this section.

3.2.5 Geometrical transformations of all the objects in the current scene

- *Get_Current_Rotation (scene, angle_x, angle_y, angle_z)*

  Goal : Return the current angles of all the objects in the scene between their initial and their present positions.
  Input : The scene.
  Output : The three angles along the X, Y and Z axis.
  Description : The function returns the *SC_XAngle*, *SC_YAngle* and *SC_ZAngle* attributes of the *scene* occurrence of the *SCENE* entity.

- *Rotate (scene, angle_x, angle_y, angle_z)*

  Goal : Rotate all the objects in the scene from their very first position .
  Input : The scene and the three angles along the X, Y and Z axis.
  Output : The scene modified.
  Description : The function updates the *SC_XAngle*, *SC_YAngle* and *SC_ZAngle* attributes of the *scene* occurrence of the *SCENE* entity with respectively *angle_x, angle_y* and *angle_z.*

- *Get_Current_Translation (scene, method, pos_x, pos_y, pos_z)*

  Goal : Return the current position of all the objects in the scene (the values returned depends on the method choosen).

Input : The scene and the translation method (best fit, absolute or relative)
Output : The three positions along the X, Y and Z axis.
Description : The function returns the *SC_XTransl*, *SC_YTransl* and *SC_ZTransl* attributes of the *scene* occurrence of the *SCENE* entity depending on the *method* choosen.

- *Translate (scene, method, pos_x, pos_y, pos_z)*

Goal : Translate all the objects in the scene from their very first position.
Input : The scene, the translation method and the three positions along the X, Y and Z axis.
Output : The scene modified.
Description : The function updates the *SC_XTransl*, *SC_YTransl* and *SC_ZTransl* attributes of the *scene* occurrence of the *SCENE* entity with respectively *pos_x, pos_y* and *pos_z,* whose values depend on the method chosen.

- *Get_Current_Scale (scene, scale_x, scale_y, scale_z)*

Goal : Return the current scaling percentage of all the objects in the scene.
Input : The scene.
Output : The three scaling percentages along the X, Y and Z axis.
Description : The function returns the *SC_XAngle*, *SC_YAngle* and *SC_ZAngle* attributes of the *scene* occurrence of the *SCENE* entity.

- *Change_Scale (scene, scale_x, scale_y, scale_z)*

Goal : Change the scale of all the objects in the scene from their very first size.
Input : The scene and the three scaling percentages along the X, Y and Z axis.
Output : The scene modified.
Description : The function updates the *SC_XAngle*, *SC_YAngle* and *SC_ZAngle* attributes of the *scene* occurrence of the *SCENE* entity with respectively *scale_x, scale_y* and *scale_z.*

- *Display (scene)*

See item "1. Creation of a new scene" of this section.

3.2.6 Cutting of a part of the current scene

- *Define_Cp (cutting_plane_name, distance_percentage, angle1, angle2)*

Goal : Define a cutting plane.
Input : The cutting plane type (top, bottom, left, right, front or back), the distance percentage from one side of the scene and the two angles between the cutting plane and the side of the scene.

Output : The cutting plane.

Description : The function creates a new occurrence of the *CUTTING_PLANE* entity, updates its *CP_Name* attribute with *cutting_plane_name*.
If *CP_Name* is "front" or "back", the *CUTTING_PLANE* entity is a *FRONT_BACK_CP* entity and the function updates its *FBCP_DepthPerc* attribute with *distance_percentage*, it updates its *FBCP_XAngle* attribute with *angle1* and it updates its *FBCP_YAngle* attribute with *angle2*.
If *CP_Name* is "left" or "right", the *CUTTING_PLANE* entity is a *LEFT_RIGHT_CP* entity and the function updates its *LRCP_DepthPerc* attribute with *distance_percentage*, it updates its *LRCP_YAngle* attribute with *angle1* and it updates its *LRCP_ZAngle* attribute with *angle2*.
If *CP_Name* is "top" or "bottom", the *CUTTING_PLANE* entity is a *TOP_BOTTOM_CP* entity and the function updates its *TBCP_DepthPerc* attribute with *distance_percentage*, it updates its *TBCP_XAngle* attribute with *angle1* and it updates its *FBCP_ZAngle* attribute with *angle2*.

- *Display_Cp (scene, cutting_plane, color, size, display_type, value)*

Goal : Display the cutting plane into the scene.

Input : The scene and the cutting plane previously defined.

Output : The scene modified.

Description : The function creates an occurrence of the *SHOWING_CP* relation between the *cutting_plane* occurrence of the *CUTTING_PLANE* entity and the *scene* occurrence of the *SCENE* entity, updates the *CP_Color* attribute with *color*, updates the *CP_Size* attribute with *size*, updates the *CP_Grid* attribute with *display_type*.
If *CP_Grid* is true, the function updates the *CP_Wires* attribute with *value* that and forget the *CP_Transparency* attribute.
If *CP_Grid* is false, the function updates the *CP_Transparency* attribute with *value* and forget the *CP_Wires* attribute.

- *Cut_Scene (scene, cutting_plane)*

Goal : Cut a part of the scene the scene.

Input : The scene and the cutting plane previously defined.

Output : The scene modified.

Description : The function creates an occurrence of the *CUTTING* relation between the *cutting_plane* occurrence of the *CUTTING_PLANE* entity and the *scene* occurrence of the *SCENE* entity.

- *Display (scene)*

See item "1. Creation of a new scene" of this section.

3.2.7 Management of the lights

- *Turn_Light_On (scene, light_id, color, specular_component, far, x, y, z)*

  Goal : Turn a light on in a scene.

  Input : The scene, the light identifier (maximum eight lights), the color, the specular component, the far paremeter. If *far* is true, the last three inputs are the light direction else they represent the light position.

  Output : The scene modified.

  Description : The function creates an occurrence of the *LIGHT* entity, updates the *LT_Id* attribute with *light_id*, updates the *LT_Color* attribute with *color*, updates the *LT_Specular* attribute with *specular_component*, updates the *LT_Far* attribute with *far*, updates the *LT_XPos* attribute with *x*, updates the *LT_YPos* attribute with *y* and updates the *LT_ZPos* attribute with *z*. It creates a *LIGHTING* relation between this occurrence of the *LIGHT* entity and the *scene* occurrence of the *SCENE* entity.

- *Turn_Light_Off (scene, light_id)*

  Goal : Turn a light of a scene off.

  Input : The scene and the light identifier (maximum eight lights).

  Output : The scene modified.

  Description : The function destroys the occurrence of the relation between the *LIGHT* entity whose *LT_Id* attribute corresponds to *light_id* and the *scene* occurrence of the *SCENE* entity. It also destroys this occurrence of the *LIGHT* entity itself.

- *Turn_All_Lights_Off (scene)*

  Goal : Turn all the lights of a scene off.

  Input : The scene.

  Output : The scene modified.

  Description : The function destroys all the occurrences of the relation between the *LIGHT* entity and the *scene* occurrence of the *SCENE* entity. It also destroys all the occurrences of the *LIGHT* entity themselves.

- *Display (scene)*

  See item "1. Creation of a new scene" of this section.

3.2.8 Saving into VRML format

- *Save_To_VRML (scene, file_name)*

  Goal : Save the scene description into a VRML format file.
  Input : The scene and the file that will contain the scene description.
  Output : /
  Description : The function translates the *scene* description into a VRML hierarchy of primitives and saves the result in the file whose name is *file_name*.

3.2.9 Addition of an object into the current scene

- *Load_Object (file_name)*

  See item "1. Creation of a new scene" of this section.

- *Add_Object (scene, object)*

  See item "1. Creation of a new scene" of this section.

- *Select_Current_Object (scene, object)*

  See item "1. Creation of a new scene" of this section.

- *Change_Name (scene, object, name)*

  See item "1. Creation of a new scene" of this section.

- *Display (scene)*

  See item "1. Creation of a new scene" of this section.

3.2.10 Selection of the current object

- *Get_Object (scene, object_name)*

  Goal : Retrieve the current object in a scene in function of his name
  Input : The scene and the object name.
  Output : The current object of the scene.
  Description : The function returns the occurrence of the *OBJECT* entity that *Is_current_object_of* the *scene* occurrence of the *SCENE* entity.

- *Select_Current_Object (scene, object)*

  See item "1. Creation of a new scene" of this section.

3.2.11 Removal of the current objet from the current scene

- *Remove_Object (scene, object)*

  Goal : Remove an object from a scene.

  Input : The scene and the object.

  Output : The scene modified and another object.

  Description : The function destroys the *object* occurrence of the *OBJECT* entity that *Is_contained_by* the *scene* occurrence of the *SCENE* entity and it destroys all the occurrences of the relations that connect this occurrence of the *OBJECT* entity to other entities. Moreover, it destroys all the occurrences of the following entities — and the occurrences of the relations between them — that are in relation with this occurrence of the *OBJECT* entity : *AXIS*, *BOX*, *TRIANGLE*, *POINT* and *TETRAHEDRON*. The function returns another occurrence of the *OBJECT* entity. The precondition to the function is the fact that there must be at least two objects in the scene.

- *Select_Current_Object (scene, object)*

  See item "1. Creation of a new scene" of this section.

- *Display (scene)*

  See item "1. Creation of a new scene" of this section.

3.2.12 Changing the name of the current object

- *Change_Name (scene, object, name)*

  See item "1. Creation of a new scene" of this section.

3.2.13 Getting information about an object

- *Get_Object (scene, object_name)*

  See item "10. Selection of the current object" of this section.

- *Get_Number_Of_Vertices (scene, object)*

  Goal : Retrieve the number of nodes of a specific object in a scene.

  Input : The scene and the object.

  Output : The number of nodes that make up the object.

  Description : The function returns the number of occurrences of the *POINT* entity that *Constitute* the occurrences of the *TETRAHEDRON* entity, themselves *Composing*

the *object* occurrence of the *OBJECT* entity and this entity *Being_contained_by* the *scene* occurrence of the *SCENE* entity.
Or it returns the number of occurrences of the *POINT* entity that *Constitute* the occurrences of the *TRIANGLE* entity, themselves *Composing* the *object* occurrence of the *OBJECT* entity and this entity *Being_contained_by* the *scene* occurrence of the *SCENE* entity.

- *Get_File_Name (scene, object)*

  Goal : Retrieve the name of the files that contain the description of a specific object in a scene.
  Input : The scene and the object.
  Output : The name of the file.
  Description : The function returns the *OBj_FileName* attribute of the *object* occurrence of the *OBJECT* entity that *Is_contained_by* the *scene* occurrence of the *SCENE* entity.

- *Get_Visualization_Type (scene, object)*

  Goal : Retrieve the type of visualization used to display a specific object in a scene.
  Input : The scene and the object.
  Output : The visualization type ("Filled Surface", "Mesh" or "Points Cloud").
  Description : The function returns the *OBj_DispType* attribute of the *object* occurrence of the *OBJECT* entity that *Is_contained_by* the *scene* occurrence of the *SCENE* entity.

- *Get_Number_Of_Triangles (scene, object)*

  Goal : Retrieve the number or elements (triangles) that compose a specific object in a scene.
  Input : The scene and the object.
  Output : The number of triangles.
  Description : The function returns the number of occurrences of the *TRIANGLES_COMP* relation between the occurrences of the *TRIANGLE* entity and the *object* occurrence of the *OBJECT* entity that *Is_contained_by* the *scene* occurrence of the *SCENE* entity.

- *Get_Number_Of_Tetrahedrons (scene, object)*

  Goal : Retrieve the number or elements (tetrahedrons) that compose a specific object in a scene.
  Input : The scene and the object.
  Output : The number of tetrahedrons.
  Description : The function returns the number of occurrences of the *TETRAHED_COMP* relation between the occurrences of the *TETRAHEDRON* entity and the *object*

occurrence of the *OBJECT* entity that *Is_contained_by* the *scene* occurrence of the *SCENE* entity.

3.2.14 Changing the color of the current object

- *Change_Color (scene, object, color, specular, shininess, translucence)*

  Goal : Change the color properties of a specific object in a scene.

  Input : The scene, the object, the color, the specular component, the shininess percentage and the translucence percentage.

  Output : The scene modified.

  Description : The function updates the following attributes of the *object* occurrence of the *OBJECT* entity that *Is_contained_by* the *scene* occurrence of the *SCENE* entity : *OBj_Color* attribute with *color*, *OBj_Specular* attribute with *specular*, *OBj_Shininess* attribute with *shininess* and *OBj_Transluc* attribute with *translucence.*

- *Display (scene)*

  See item "1. Creation of a new scene" of this section.

3.2.15 Changing the type of visualization of the current object

- *Change_Visualization_Type (scene, object, visualization_type)*

  Goal : Change type of display of a specific object in a scene.

  Input : The scene, the object and the type of display ("Filled Surface", "Mesh" or "Points Cloud").

  Output : The scene modified.

  Description : The function updates the *OBj_DispType* attribute of the *object* occurrence of the *OBJECT* entity that *Is_contained_by* the *scene* occurrence of the *SCENE* entity with *visualization_type.*

- *Display (scene)*

  See item "1. Creation of a new scene" of this section.

3.2.16 Showing the current object axis

- *Show_Axis_On (scene, object , color, length, move)*

  Goal : Display an axis with a specific object in a scene.

Input : The scene, the object, the axis color, the axis length and a parameter that tell if the axis move or not with the object.

Output : The scene modified.

Description : The function creates an occurrence of the *AXIS* entity and an occurrence of the *SHOWING_AXIS* relation that connects the previous entity with the *object* occurrence of the *OBJECT* entity that *Is_contained_by* the *scene* occurrence of the *SCENE* entity. It updates the following attributes of the occurrence of the *AXIS* entity : *Axis_Color* with *color*, *Axis_Length* with *length* and *Axis_Move* with *move*.

- *Show_Axis_Off (scene, object)*

Goal : Display a specific object in a scene without its axis.

Input : The scene and the object.

Output : The scene modified.

Description : The function destroys the occurrence of the *SHOWING_AXIS* relation that connects the *object* occurrence of the *OBJECT* entity that *Is_contained_by* the *scene* occurrence of the *SCENE* entity with the occurrence of the *AXIS* entity. Moreover, it destroys the occurrence of the *AXIS* entity.

- *Display (scene)*

See item "1. Creation of a new scene" of this section.

3.2.17 Showing the current object box

- *Show_Box_On (scene, object, color, size)*

Goal : Display a box with a specific object in a scene.

Input : The scene, the object, the box color, and the box size.

Output : The scene modified.

Description : The function creates an occurrence of the *BOX* entity and an occurrence of the *SHOWING_BOX* relation that connects the previous entity with the *object* occurrence of the *OBJECT* entity that *Is_contained_by* the *scene* occurrence of the *SCENE* entity. It updates the following attributes of the occurrence of the *BOX* entity : *Box_Color* with *color* and *Box_Size* with *size*.

- *Show_Box_Off (scene, object)*

Goal : Display a specific object in a scene without its box.

Input : The scene and the object.

Output : The scene modified.

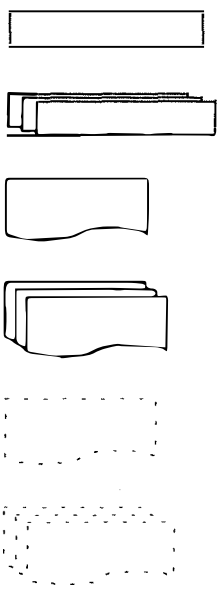
Description : The function destroys the occurrence of the *SHOWING_BOX* relation that connects the *object* occurrence of the *OBJECT* entity that *Is_contained_by* the *scene* occurrence of the *SCENE* entity with the occurrence of the *BOX* entity. Moreover, it destroys the occurrence of the *BOX* entity.

- *Display (scene)*
  See item "1. Creation of a new scene" of this section.

3.3 Composition of the Activity Chaining Graph (ACG)

In this section is described the dynamic aspect of the task with an activity chaining graph which "*expresses the information flow between functions to be executed for achieving the main goal associated with an interactive task*" as explained in detail in [BODART95a]. To simplify and because we see our application as a toolbox, an activity chaining graph is constructed for each elementary sub-task.

The syntax legend of such a graph is explained in Figure 4-7.



Function (computer processing).

Function with multiple triggering.

External information in input or output of a function.

External information in multiple occurences

Internal information in input or output of a function.

Repeated internal information

No link — OR.

(— AND.

((— Exclusive OR (XOR).

Figure 4-7 : Graphical conventions for ACG

The rectangles represent semantic functions, that is to say functions that carry out a process. In 2D or 3D imaging applications, some functions do not realize a process but are simply used to display a result. These functions are called *service functions* and "Display (scene)" is an example of such a function. The problem we faced, was to decide how to represent the service functions. Two possibilities exist. Either we do not represent them explicitly but their result — the display — are visualized by an external information or we represent them by a rectangle — like the semantic functions — and their result by an external information. We decided to used the second option because we think that service functions are central elements in visualization applications.

The ACG model used in this text is different from the one usually used in the TRIDENT methodology in the sense that it accepts messages to be the result of several functions. In Figure 4-8, both functions a and b are parallel. The message obtained is either the result of only one of them or the result constructed at the same time by both functions.

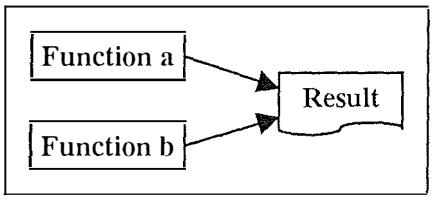


Figure 4-8 : Parallel functions

In most of the activity chaining graphs you will see the internal message *Current_Scene* and some other messages *Current_Scene'*, *Current_Scene''*... The first one represents a description of the scene currently used. The others represent the same scene but modified. So, *Current_Scene''* is *Current_Scene'* modified and *Current_Scene'* is *Current_Scene* modified. If *Current_Scene''* is the last version of the current scene modified in an ACG, then it becomes *Current_Scene* — the current scene in use — in another ACG. The same explanation stands for *Current_Object* and *Current_Object'*, *Current_Object''*...

The internal messages *Current_Scene*, *Current_Scene'*, *Current_Scene''*... are logical descriptions of the current scene. We mean that they represent the data structure of the current scene. On the other hand, the external message *Scene* that is the output of the *Display* function represents the physical description of the scene, that is to say what the users see on the screen.

3.3.1 Creation of a new scene

Figure 4-9 is showing the ACG for the sub-task "Creation of a new scene".

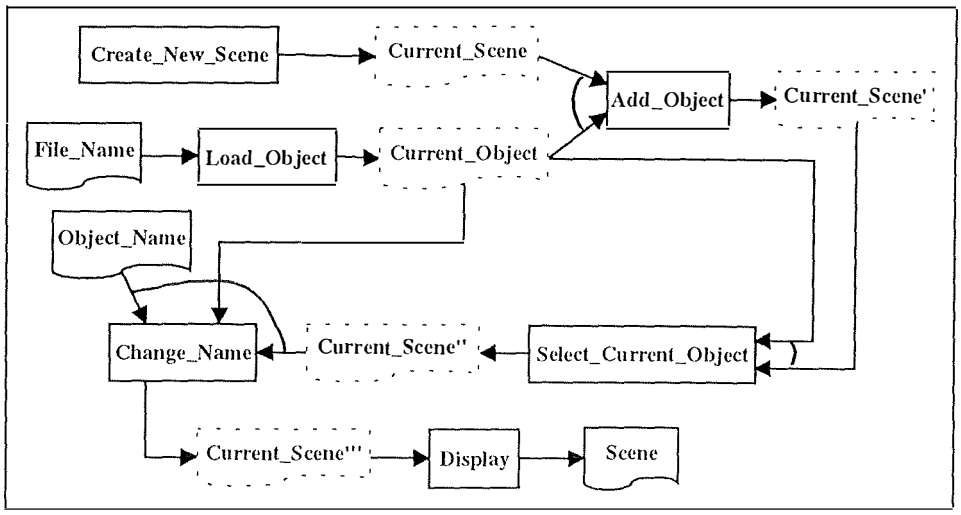


Figure 4-9 : ACG "Creation of a new scene"

3.3.2 Selection of the current scene

Figure 4-10 is showing the ACG for the sub-task "Selection of the current scene".

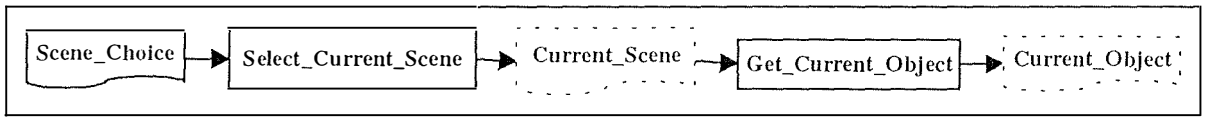


Figure 4-10 : ACG "Selection of the current scene"

3.3.3 Removal of the current scene

Figure 4-11 is showing the ACG for the sub-task "Removing of the current scene".

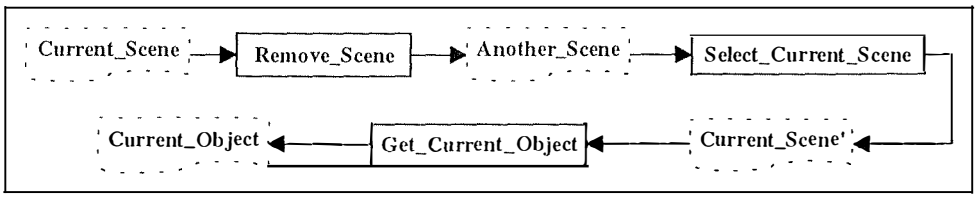


Figure 4-11 : ACG "Removing of the current scene"

3.3.4 Specifying the parameters of the current scene

Figure 4-12 is showing the ACG for the sub-task "Changing the parameters of the scene".

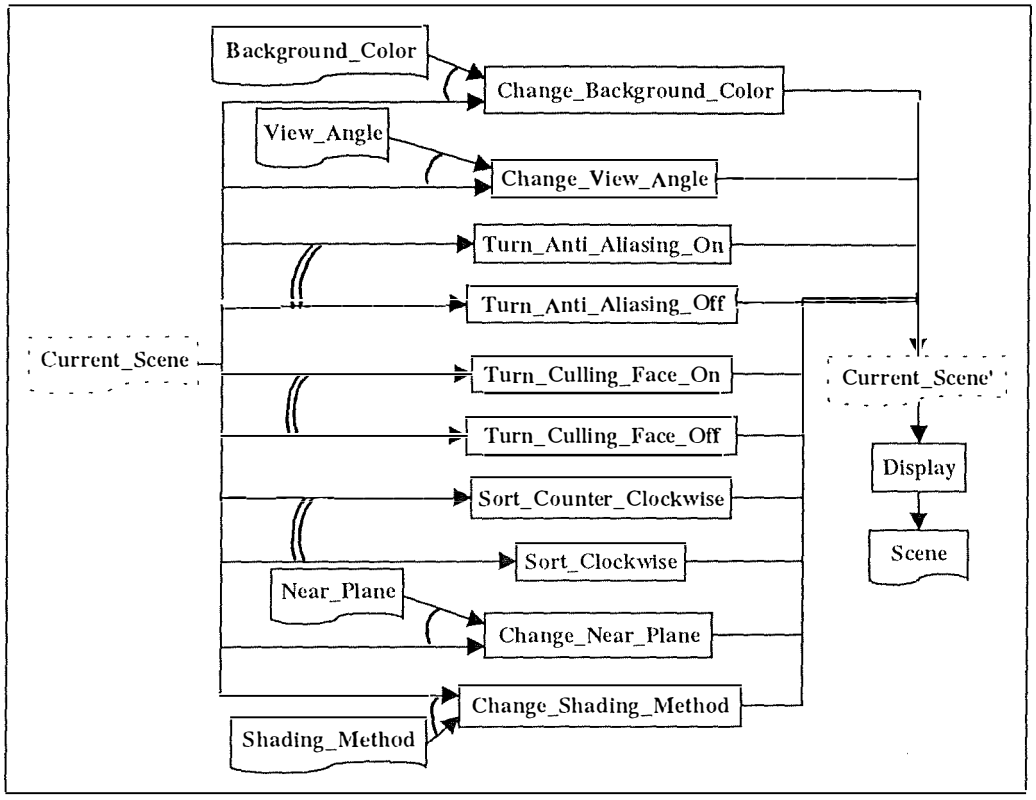


Figure 4-12 : ACG "Changing the parameters of the scene"

3.3.5 Geometrical transformation of all the objects in the current scene

Figure 4-13 is showing the ACG for the sub-task "Geometrical transformation of all the objects in the scene".

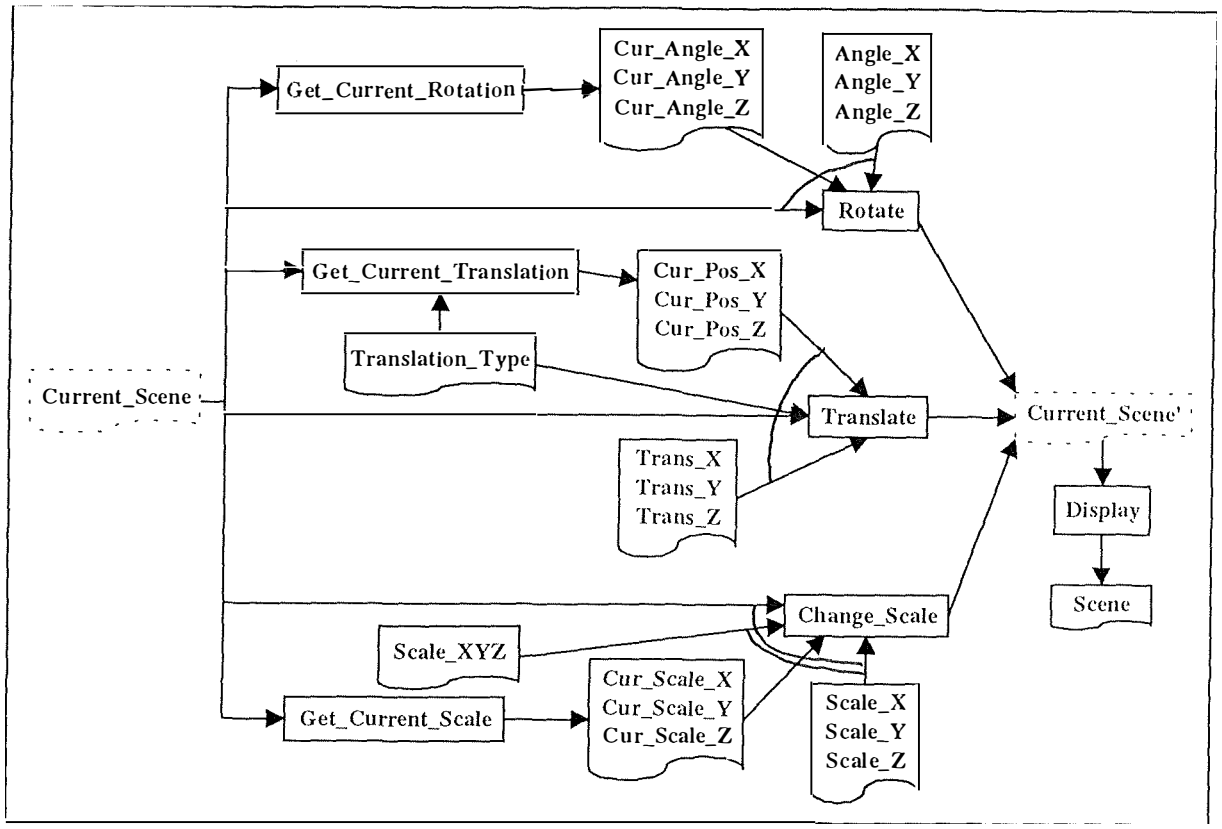


Figure 4-13 : ACG "Geometrical transformation of all the objects in the scene"

3.3.6 Cutting a part of the current scene

Figure 4-14 is showing the ACG for the sub-task "Cutting a part of the scene". We will explain more in depth the four inputs for the function "define_cp". Parameter "Scene_type" specify which of the six cutting planes has to be used. Therefore it can take up to six different values : "*TOP*", "*BOTTOM*", "*LEFT*", "*RIGHT*", *BACK*" and "*FRONT*". The meaning of the other three parameters depends on which value has been chosen. We will review each of the six possible values and explain what are the meaning of the other parameters.

If the Scene_type value is "TOP", the parameter "distance" means the distance percentage between the top cutting plane and the top of the scene. In this case, angle_1 represents the angle between this cutting plane and the top of the scene along the x axis and angle_2 represents the angle between them along the z axis. If the Scene_type value is "BOTTOM", the three parameters have the same meaning as above but they concern the bottom cutting plane and the bottom side of the scene.

If the Scene_type value is "LEFT", the parameter "distance" means the distance percentage between the left cutting plane and the left boundary of the scene. In this case, angle_1 represents

the angle between this cutting plane and the left boundary of the scene along the y axis and angle_2 represents the angle between them along the z axis. If the Scene_type value is "RIGHT", the three parameters have the same meaning as above but they concern the right cutting plane and the right boundary of the scene.

If the Scene_type value is "BACK", the parameter "distance" means the distance percentage between the back cutting plane and the back boundary of the scene. In this case, angle_1 represents the angle between this cutting plane and the back boundary of the scene along the x axis and angle_2 represents the angle between them along the y axis. If the Scene_type value is "FRONT", the three parameters have the same meaning as above but they concern the front cutting plane and the front boundary of the scene.

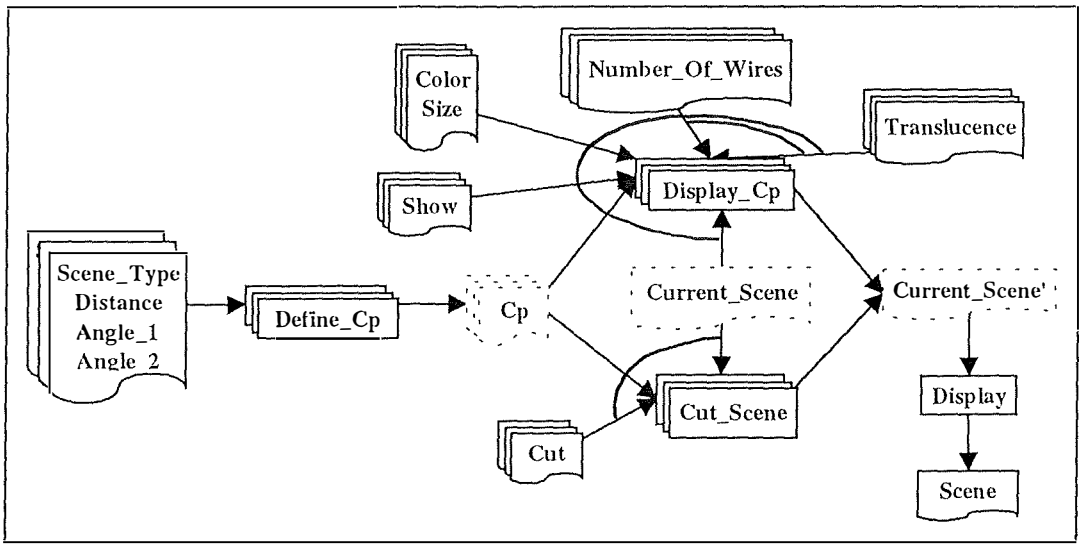


Figure 4-14 : ACG "Cutting a part of the scene"

3.3.7 Management of the lights

Figure 4-15 is showing the ACG for the sub-task "Management of the lights".

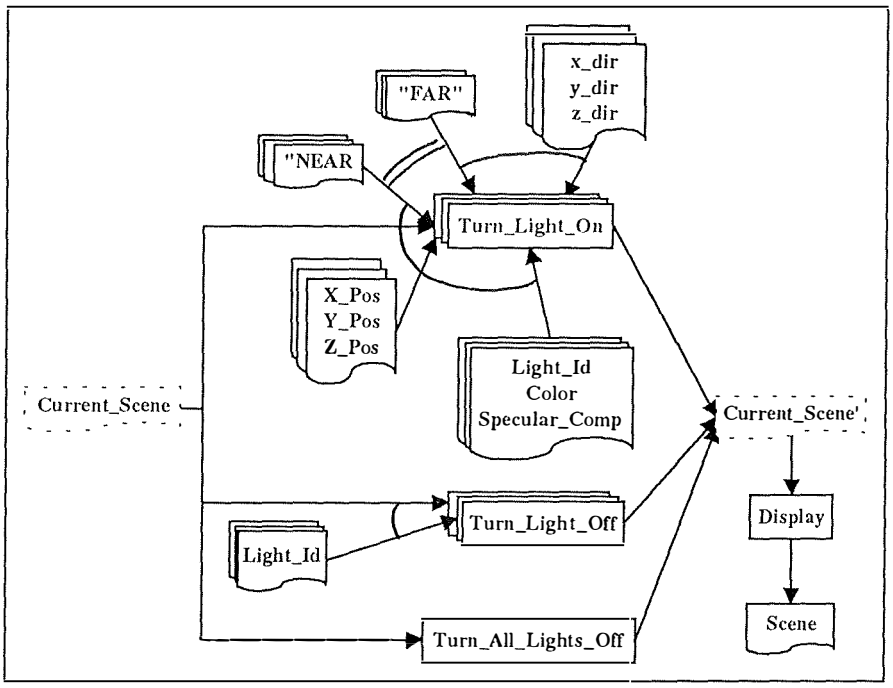


Figure 4-15 : ACG "Management of the lights"

3.3.8 Saving into VRML format

Figure 4-16 is showing the ACG for the sub-task "Saving into VRML format".

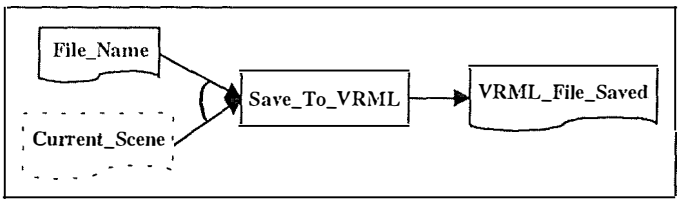


Figure 4-16 : ACG "Saving into VRML format"

3.3.9 Addition of an object into the current scene

Figure 4-17 is showing the ACG for the sub-task "Addition of an object into the current scene".

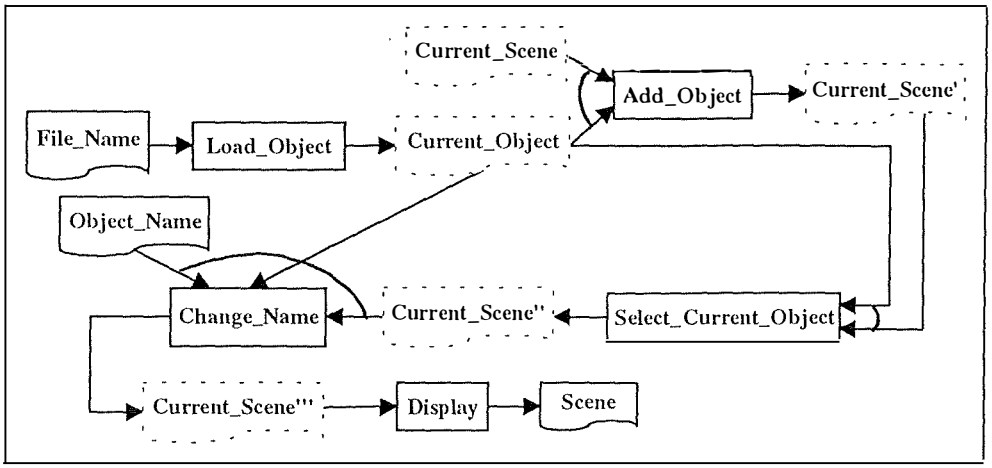


Figure 4-17 : ACG "Addition of an object into the current scene"

3.3.10 Selection of the current object

Figure 4-18 is showing the ACG for the sub-task "Selection of the current object".

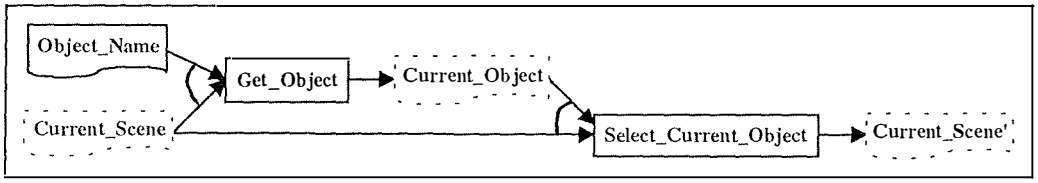


Figure 4-18 : ACG "Selection of the current object"

3.3.11 Removal of the current object from the current scene

Figure 4-19 is showing the ACG for the sub-task "Removal of the current object from the current scene".

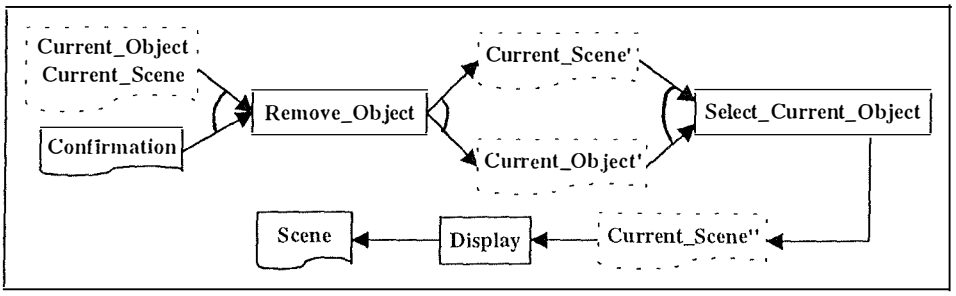


Figure 4-19 : ACG "Removal of the current object from the current scene"

3.3.12 Changing the name of the current object

Figure 4-20 is showing the ACG for the sub-task "Changing the name of the current object".

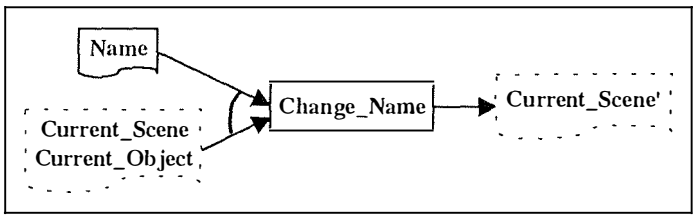


Figure 4-20 : ACG "Changing the name of the current object"

3.3.13 Getting information about an object

Figure 4-21 is showing the ACG for the sub-task "Getting information about an object".

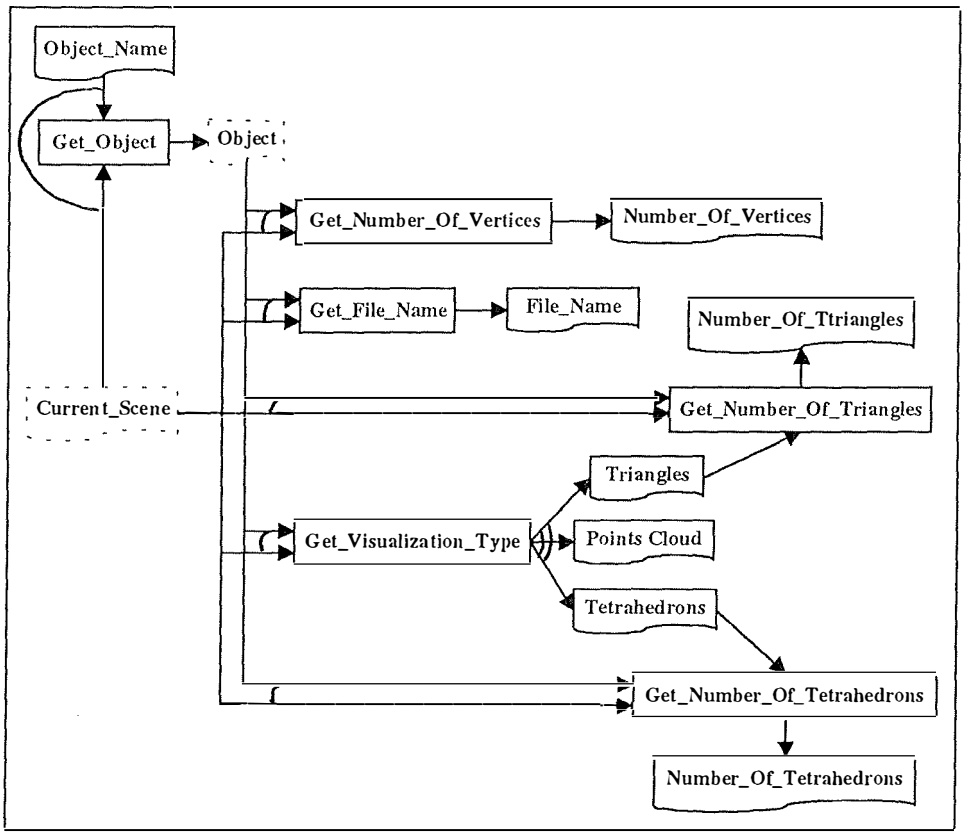


Figure 4-21 : ACG "Getting information about an object"

3.3.14 Changing the color of the current object

Figure 4-22 is showing the ACG for the sub-task "Changing the color of the current object".

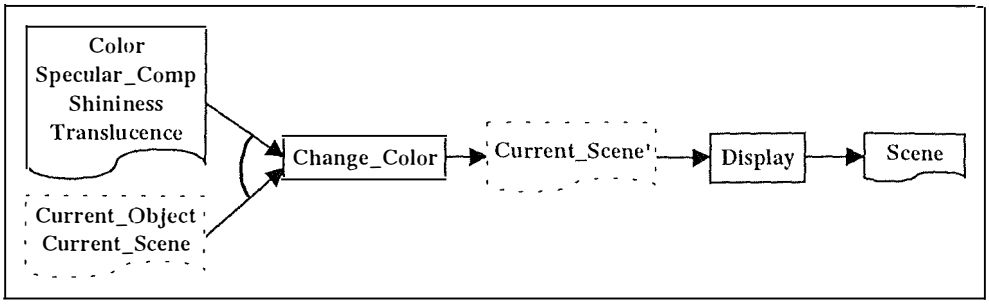


Figure 4-22 : ACG "Changing the color of the current object"

3.3.15 Changing the type of visualization of the current object

Figure 4-23 is showing the ACG for the sub-task "Changing the type of visualization of the current object".

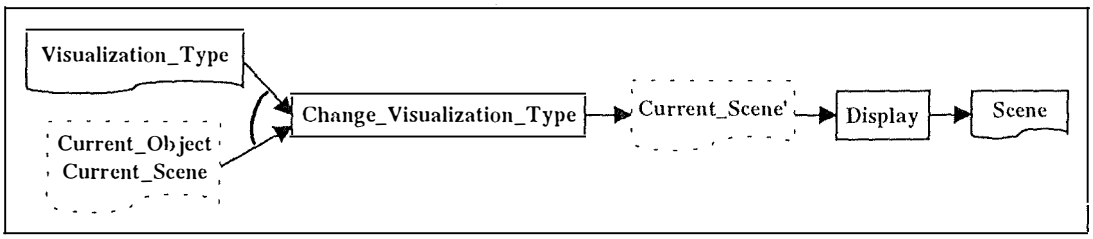


Figure 4-23 : ACG "Changing the type of visualization of the current object"

3.3.16 Showing the current object axis

Figure 4-24 is showing the ACG for the sub-task "Showing the current object axis".

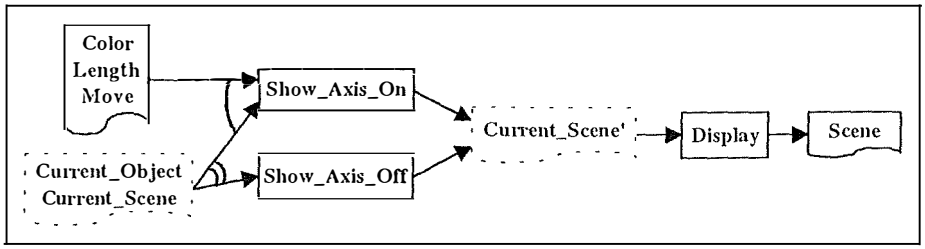


Figure 4-24 : ACG "Showing the current object axis"

3.3.17 Showing the current object box

Figure 4-25 is showing the ACG for the sub-task "Showing the current object box".

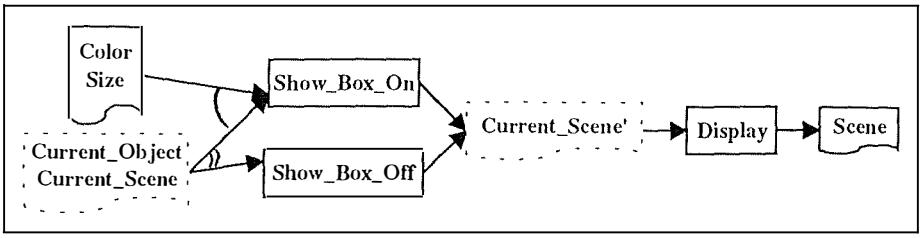


Figure 4-25 : ACG "Showing the current object box"

3.4 Derivation of dialogue attributes

Five dialogue attributes will be analyzed as explained in [VANDERDONCKT93a] : (1) the dialogue control, (2) the dialogue sequencing, (3) the dialogue mode, (4) the functions triggering mode and (5) the metaphor.

The dialogue attributes are usually derived from a simple interaction style or several interaction styles combined together, themselves being determined with the help of tables of correspondence between the parameters relative to the task, the users stereotypes, the working environment and the interaction styles advocated. Before going on with the specification of the dialogue attributes, we will therefore pick up one or several suitable interaction styles for our application 3D Viewer with the tables of correspondence defined in [VANDERDONCKT93a].

3.4.1 Interaction styles derivation

	Prerequisite low	Productivity moderate	Objective environment existent	Environment reproductibility practicable	Task structuration low	Task importance high	Task complexity moderate
Natural language	X				X		X
Command language	X				X	X	X
Interrogation language	X	X			X		
Questions/ answers	X		X	X			
Function keys	X				X		X

Menu selection	X	X					X
Form filling	X	X	X	X		X	X
Multi-windows	X	X	X	X	X	X	
Direct manipulation	X	X	X	X	X	X	
Iconic interaction	X		X	X		X	X

Table 4-1 : Parameters relative to the interactive task.

	Task experience elementary	Systems experience rich	Motivation high	Complex interaction means experience rich
Natural language				X
Command language		X	X	X
Interrogation language				X
Questions/ answers	X			X
Function keys	X			
Menu selection				
Form filling				
Multi-windows				
Direct manipulation				
Iconic interaction				

Table 4-2 : Parameters relative to the users stereotypes.

	Processing type multi-processing	**Processing capacity** moderate to high
Natural language		
Command language		
Interrogation language		
Questions/ answers		
Function keys	X	X
Menu selection	X	X
Form filling	X	X
Multi-windows	X	X
Direct manipulation	X	X
Iconic interaction	X	X

Table 4-3 : Parameters relative to the workplace.

The three interaction styles derived from the analysis of the tables of correspondence (Table 4-1, Table 4-2 and Table 4-3) that are the best to use in our case are the *multi-windows*, the *form filling* and the *direct manipulation.* However, during the implementation of the software, we have chosen, besides the quoted interaction styles, the *menu selection.* These choices will be justified below.

The menu selection suits very well for cases where the application is the metaphor of a toolbox. No determined process exists. The users create their own process ad-hoc during the use of the program. The menu selection represents the selection of tools in the toolbox.

The forms filling corresponds to the parameters settings of the tools used. For example, if the user decides to work with lights — a tool —, he will be asked to select the lights he wants to use and to determine their characteristic (color components, position, shininess...).

The multi-windows interaction style is used to show different views of the same scene or to show different scenes at the same time. The advantage of multi-view has been discussed in *Chapter 2 :*

Description of the process for the visualization program. As for the direct manipulation, it will be used to rotate all the object of a scene, but is not implemented yet. We are now going to derive the dialogue attributes.

3.4.2 Dialogue control

This attribute answers to the question "*who controls the dialogue ?*". The dialogue control is **variable**. In general it is internal. That is the case, for example, for the selection of menus where only the allowed menu items are proposed to the user. The other menu items are automatically grayed by the application. Likewise it is internal, inside a sub-task. Nevertheless it is external when the user goes from a window to another, each one containing a view of the same scene or a different scene.

3.4.3 Dialogue sequencing

This attribute answers to the question "*how many dialogues is it possible to control at the same time ?*". The dialogue sequencing is for the most part **mono-thread hierarchic** and sometimes it is **multi-thread multi-programmed**. Indeed, the dialogue sequencing is mono-thread hierarchic because the actions are organized hierarchically and only a part of the actions accessible at some point are made available by the interface. It is the case for the menu selection and the dialog boxes selection. However we are also considering that the sequencing can be multi-thread multi-programmed when several windows of the same scene or different scenes are displayed on the screen. When an object in a scene is rotating in a way and at a certain rotation speed, at the same time, another object could rotate in another window, in another way and at another rotation speed.

3.4.4 Dialogue mode

This attribute answers to the question "*how are the dialogues controlled ?*". The dialogue mode is **asynchronous**. Inside each sub-task, the order of actions execution and the order of data capture are not determined. As for some sub-tasks, the dialogue mode is sequential. For example, before changing the name of an object, the latter has to be loaded into a scene (cf. Figure 4-20). At the interactive task level, the dialogue mode is also asynchronous as suggested by the toolbox model.

3.4.5 Functions triggering mode

This attribute answers to the question "*who triggers the functions ?*". We remember that we distinguished two types of functions : the semantic functions and the service ones. Whatever their

type, the functions triggering mode is **displayed explicit manual**. There will always be a command button or an icon on the toolbar that will permit to trigger such a function or such other function. The goal is to give the user the possibility to control the progress of the task because they are experts. Moreover, the kind of task based on the toolbox model suggests this function triggering mode. We point out that some functions will have an automatic manual triggering mode as it is the case with the selection of the current object after being loaded into a scene.

3.4.6 Metaphor

This attribute answers to the question "*what the application is the metaphor of ?*". The metaphor is **mixed**. On the one hand it is based on the conversation when the user must fill in the forms, when he has to choose the sub-task to carry out... On the other hand it is based on the universe when speaking of the representation of a scene on the computer screen where real objects (pieces of bones, muscles, fat...) are shown, where cutting planes are visible, where lights have visible results...

4. Conclusion

4.1 Critic

We remind you that we did not observe the task and that we know nothing about the mental process of posing a medical diagnosis, what the application is the support of. Our approach was first the design of a prototype of a 3D visualization application without any methodology and then a validation of the prototype with the TRIDENT methodology. Nevertheless we have found the following task characteristics :

- Prescribed (not observed).
- Decision support.

As a consequence, the task is weakly structured and no process is determined. It is the reason why the application that helps to carry out the task is based on the toolbox metaphor (Cf. Figure 4-1 : The toolbox metaphor.). The user uses the tool he wants in function of the result he wants; nothing is imposed.

Accordingly, we have chosen — before using the TRIDENT methodology — the following interaction styles :

- Menu selection → tool selection.
- Form filling → parameters setting of the tool.
- Multi-windows → displaying of several scenes.
- Direct manipulation → direct manipulation of the scene.

The last interaction style is not implemented yet. We point out that the interaction styles were derived only in function of the sort of task and the imposed work environment (Windows 95 and NT), not in function of the users stereotypes.

In the thesis we applied the first dimension of the TRIDENT methodology as a validation mean of the interaction styles we have chosen. We have derived the following interaction styles from the task "analysis" suggested by the TRIDENT methodological framework (which takes into account the users stereotypes who are considered as experts) :

- Form filling
- Multi-windows
- Direct manipulation

Here, because the task is prescribed, "analysis" rather means "supposition".

Since most of the interaction styles we have chosen are the same as the one derived from the task analysis, is the task analysis still useful in the case of weakly structured tasks ? To answer to this question we are going to see if we derive the same interaction styles from the task analysis by considering users who are **beginners** in using computers and who are **not** a lot **motivated**. Table 4-4 is showing the interaction styles derived in function of the users stereotypes

By taking into account Table 4-1, Table 4-2, Table 4-3 and Table 4-4, we notice that the suitable interaction styles proposed by the task analysis are

- Form filling
- Multi-windows
- Direct manipulation

That is the same as the one when the users are experts. Whatever the users, does the weakly structured sort of task suggest the use of the form filling, multi-windows and direct manipulation interaction styles ? The answer to this question should be validated on a greater number of applications.

	Task experience elementary	**Systems experience** elementary	**Motivation** weak	**Complex interaction means experience** elementary
Natural language			X	
Command language				
Interrogation language				
Questions/ answers	X	X	X	
Function keys	X	X	X	
Menu selection		X	X	X
Form filling		X	X	X
Multi-windows		X	X	X
Direct manipulation			X	X
Iconic interaction			X	X

Table 4-4 : Parameters relative to the users stereotypes

Is the menu selection a good choice ? We consider that, seen the toolbox model, the application is assimilated to the toolbox and the menu selection allows the user to select a tool in the toolbox. Again, more analysis is necessary to be able to answer to the question.

4.2 TRIDENT methodology enrichment

The TRIDENT methodology does not take charge of weakly structured tasks (prescribed and/or decision support). So when we are facing such a task, we advise not to apply the task analysis as suggested by the methodology. We are proposing an approach that is based on the TRIDENT methodology, that takes into account the weakly structured type of task and that ends up with the ACG.

1. Decompose the task into goals and sub-goals. The result is the same hierarchy of goals and sub-goals as obtained with the TRIDENT methodology.

2. Consider the application that will help to carry out the task as a toolbox. So, in the preceding hierarchy, identify the tools and the central element on which the tools are applied. The rule of tools identification is "*one tool by sub-task of the interactive task*". In our case, the seventeen tools peculiar to each sub-task are :
   - The tool of a new scene creation
   - The tool of the current scene selection
   - The tool of the current scene removal
   - The tool of the current scene parameters setting.
   - The tool of geometrical transformations of all the objects in the current scene
   - The tool of cutting a part of the current scene
   - The tool of management of the lights
   - The tool of saving into VRML format
   - The tool of a new object addition into the current scene
   - The tool of the current object selection
   - The tool of the current objet removal from the current scene
   - The tool of the current object name changing
   - The tool of getting information about an object
   - The tool of the current object color changing
   - The tool of the current object type of visualization changing
   - The tool of the current object axis showing
   - The tool of the current object box showing

   The central elements identified are the scenes containing 3D objects (i.e. pieces of trabecular bones...).

3. Identify the procedures as in the TRIDENT methodology. A procedure is a "*combination of actions on objects that results into a particular state of the activity domain*", [BODART95a]. In the case of prescribed task, actions can directly represent semantic functions, that is to say no abstraction is performed because there is no task analysis and when elaborating the procedures we directly think in term of computer functions (Cf. step 5).

4. Identify the objects of the task from the decomposition into procedures (Cf. the TRIDENT methodology) and establish the ERA model.

5. Identify the semantic and the service (help, display... : see Chapter 4, Section 3.3) functions that are abstractions of the actions contained in the procedures identified above. When the task is prescribed, it is possible that actions are directly mapped into functions — that is to

say there is no abstraction mechanism — because we directly consider the actions as computer functions during the procedures identification.

6. Compose the ACGs — the rule : "*one ACG by tool identified in the goals and sub-goals hierarchy*" — that use the semantic and service functions identified above. The ACGs are the same as the one used in the TRIDENT methodology except that it is possible to represent parallel functions (see Figure 4-8) that work towards a common result. It should be possible to aggregate all the ACGs into only one ACG. The ACGs will be used for the presentation unit and the windows identification (see Chapter 5) and for the construction of the functional hierarchy (see Chapter 6).

7. We suggest to use of the following interaction styles :
   - Menu selection → tool selection.
   - Form filling → parameters setting of the tools.
   - Multi-windows → displaying of several windows, each one containing the central element on which the tools are applied. In our application, the central elements are the scenes; in word processor applications, the central elements are the text documents.
   - Direct manipulation → direct modification of the central elements contained in the windows. For example, in 3D viewer, the direct manipulation should consist in rotating all the objects in a scene; in a word processor, the direct manipulation is used for selecting a string.

   We insist on the fact that the interaction styles proposed are only a suggestion that should be validated on a big number of applications helping to perform weakly structured tasks.

8. Derive the dialogue attributes.

Chapter 5
Second dimension :
Presentation Design From Ergonomic Rules

1. Introduction

The presentation design will be done by following the systematic approach explained in [BODART95a] and summarized in Figure 5-1. This approach uses the following concepts : concrete interaction object (CIO), abstract interaction object (AIO), window, presentation unit (PU). The particularity of the TRIDENT methodology is the **continuity** between each step of the development process. Here, the continuity is materialized by the use of the activity chaining graphs and the user interfaces requirements defined at previous dimension. Moreover, the design of the presentation will be guided by ergonomic rules.

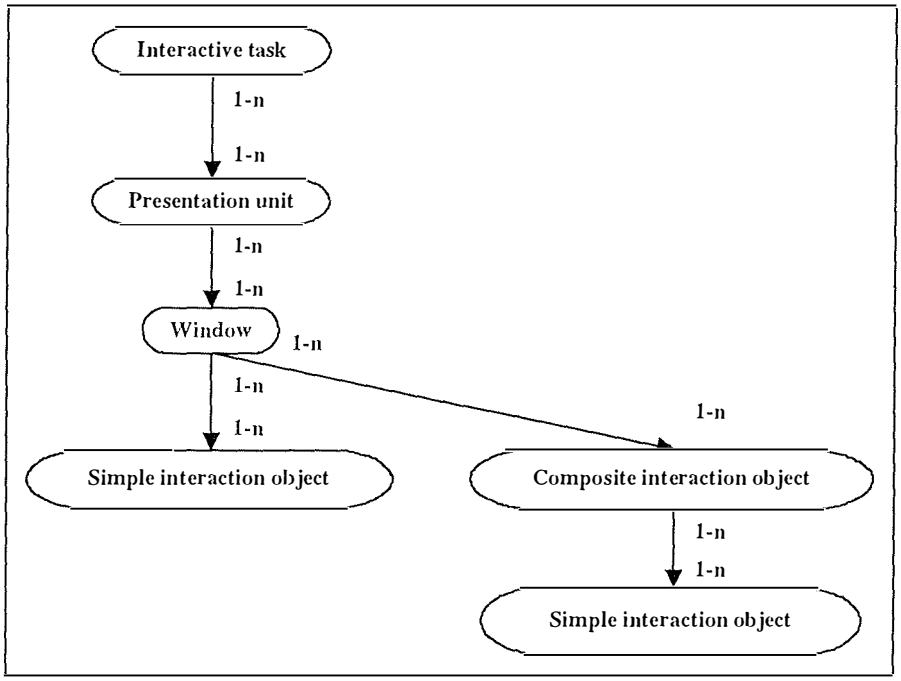


Figure 5-1 : Structure of the presentation

We will design the presentation by following the five steps suggested by the systematic approach : (1) the identification of the PUs, (2) the identification of the windows, (3) the selection of AIOs, (4) the transformation of the AIOs into CIOs and finally (5) the CIOs placement and the manual edition of the presentation.

2. PU identification

As explained in [BODART95a], "*each sub-task of the interactive task is mapped into a presentation unit*". Each sub-task of the interactive task corresponds in fact to a particular tool in the toolbox model. So, in our case, each tool of the toolbox is mapped into a presentation unit. We have identified the following presentation units :

- PU 1 → Creation of a new scene.
- PU 2 → Selection of the current scene.
- PU 3 → Removal of the current scene.
- PU 4 → Specifying the parameters of the current scene.
- PU 5 → Geometrical transformation of all the objects in the current scene.
- PU 6 → Cutting a part of the current scene.

- PU 7 → Management of the lights.
- PU 8 → Saving into VRML format.
- PU 9 → Addition of an object into the current scene.
- PU 10 → Selection of the current object.
- PU 11 → Removal of the current object from the current scene.
- PU 12 → Changing the name of the current object.
- PU 13 → Getting information about an object.
- PU 14 → Changing the color of the current object.
- PU 15 → Changing the type of visualization of the current object.
- PU 16 → Showing the current object axis and/or box.

We point out that the two sub-tasks "showing the current object axis" and "showing the current object bounding box", materialized by two different chaining graphs, will be gathered into a common presentation unit (PU 16). We do so because they play the same role : they help the user to better understand the size and position of the objects.

3. Windows identification

The windows identification approach, discussed in [BODART95a] and [TAES94], consists in having each sub-graph materializing a presentation unit corresponding to a partition of this sub-graph. These partitions correspond to windows. For the sixteen presentation units identified in the previous section, we will identify their windows.

None of the five criteria (minimal, maximal, input/outpout, functional and free) examined in [TAES94] are used to identify the windows inside each presentation unit.
Instead, an elimination method is prefered and consists in three steps. First of all, if the presentation unit is containing the following three elements linked together by arrows (see Figure 5-2), they will be grouped together inside the same window W0 :

- The hidden message "*Current_scene*"…""
- The function "*Display*"
- The visible message "*Scene*"

The identification of this window is justified by the fact that each time the scene is modified (for example an object color has changed) it must be displayed. The next step consists in identifying standard windows ("open file" window or "save file" window) that encloses external messages representing file names. It is the case for the windows W1-2 in PU 1, W8-1 in PU 8 and W9-1 in

PU 9 (see below). The identification of these windows is justified by the software engineering rule of **reusability** and by the ergonomic rule of **inter-applications coherence**. In the final step, all the external messages that are left are gathered in only one window as long as the window is not to much overloaded.

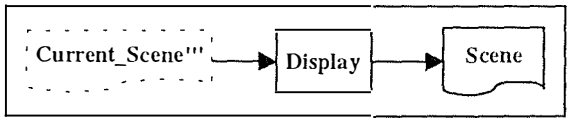


Figure 5-2 : Window W0

3.1 Windows identification for the PU 1

We have identified the following logical windows concerning the PU 1 (creation of a new scene) :

- W1-1 → Open_object_file_1
- W1-2 → Change_current_object_name_1
- W0 → Display_scene

Figure 5-3 is showing the identification of the windows contained in the first presentation unit. The window W1-1 is a standard one already defined in a Windows 95 library.

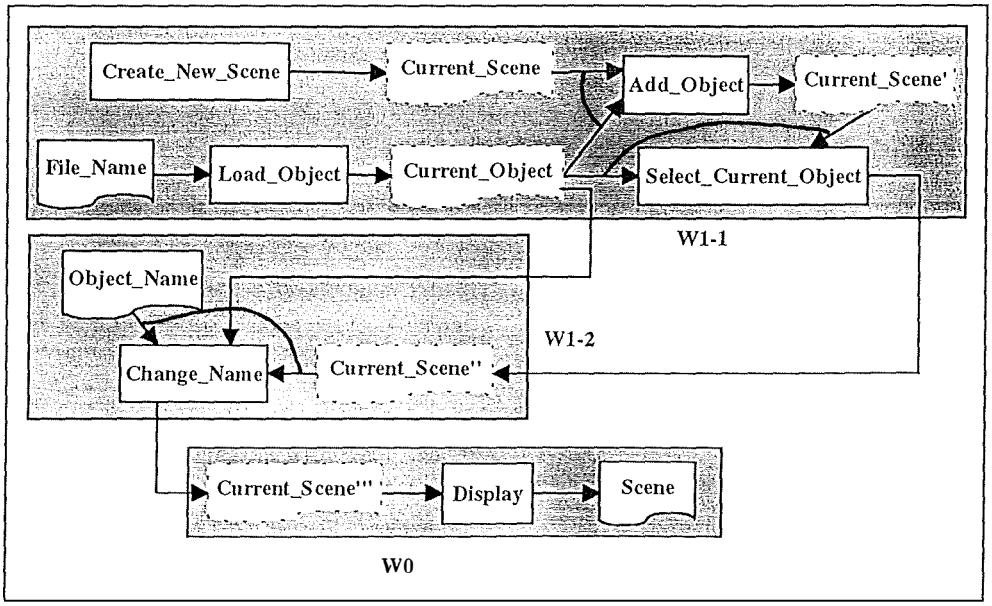


Figure 5-3 : Windows identification for the PU 1

3.2 Windows identification for the PU 2

We have identified the following logical window concerning the PU 2 (selection of the current scene) :

- W2-1 → Select_current_scene

Figure 5-4 is showing the identification of the window contained in the second presentation unit.

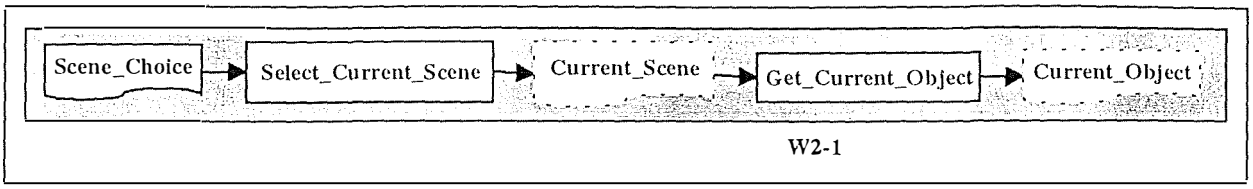


Figure 5-4 : Windows identification for the PU 2

3.3 Windows identification for the PU 3

We have identified the following logical window concerning the PU 3 (removal of the current scene) :

- W3-1 → Remove_current_scene

Figure 5-5 is showing the identification of the window contained in the third presentation unit. As you can observe, there is no external message.

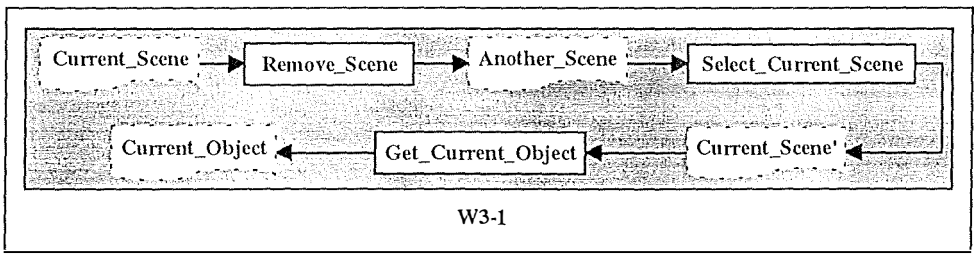


Figure 5-5 : Windows identification for the PU 3

3.4 Windows identification for the PU 4

We have identified the following logical windows concerning the PU 4 (specifying the parameters of the current scene) :

- W4-1 → Choose_scene_parameters
- W0 → Display_scene

Figure 5-6 is showing the identification of the windows contained in the fourth presentation unit.

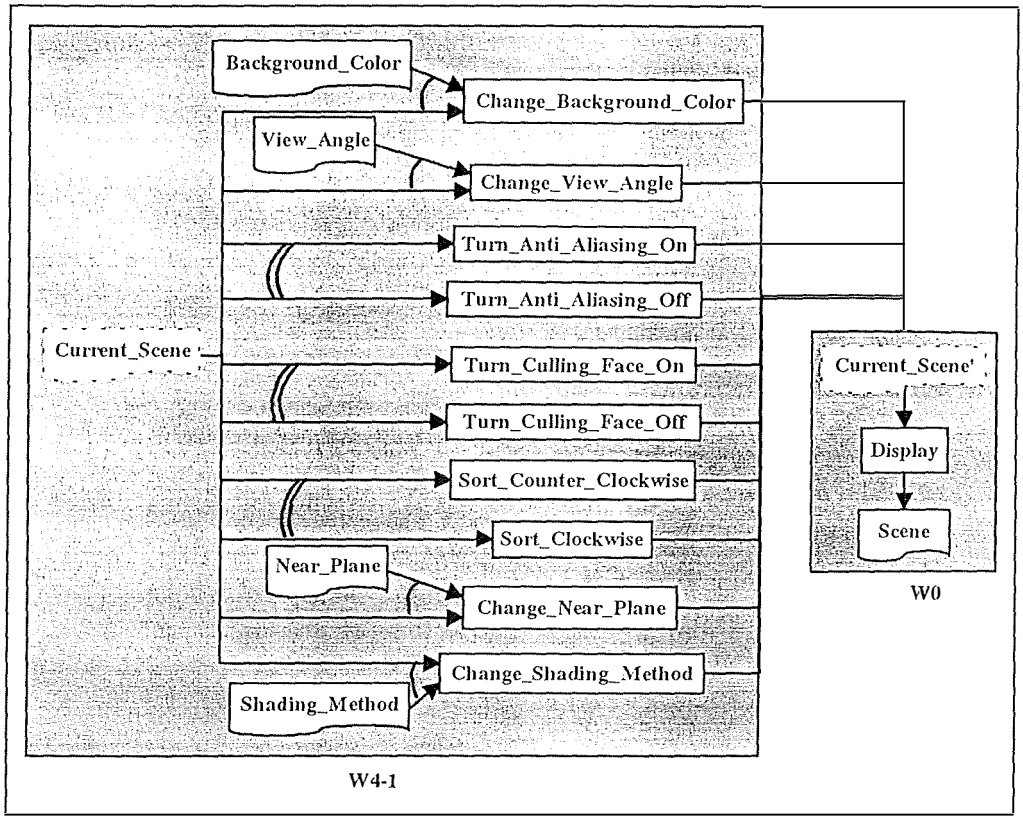


Figure 5-6 : Windows identification for the PU 4

3.5 Windows identification for the PU 5

We have identified the following logical windows concerning the PU 5 (geometrical transformation of all the objects in the current scene) :

- W5-1 → Rotate_scene
- W5-2 → Translate_scene
- W5-3 → Scale_scene
- W0 → Display_scene

Figure 5-7 is showing the identification of the windows contained in the fifth presentation unit. Even if the users are experts, we did not want to overload the window too much. It is the reason why we decided to use three windows, each one for a particular geometrical transformation, instead of only one for all the transformations.

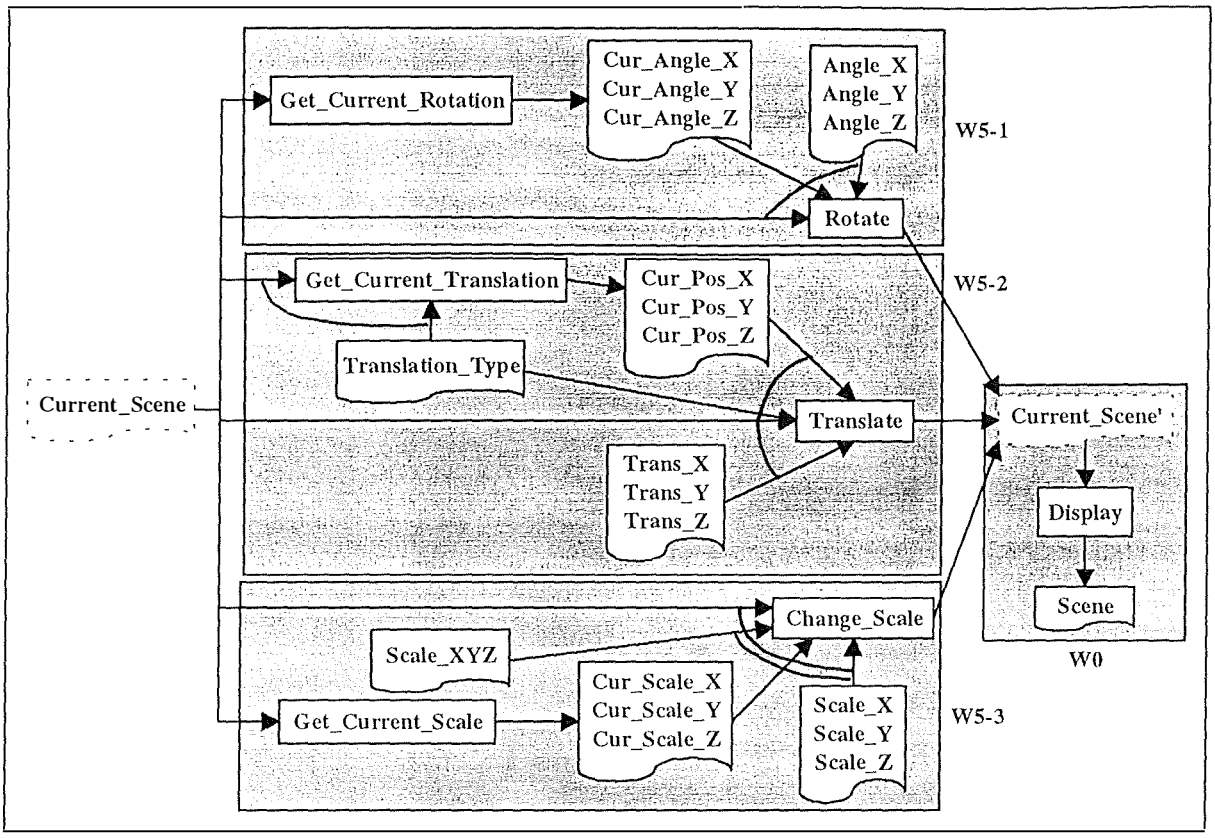


Figure 5-7 : Windows identification for the PU 5

3.6 Windows identification for the PU 6

We have identified the following logical windows concerning the PU 6 (cutting a part of the current scene) :

- W6-1 → Define_cutting_planes
- W0 → Display_scene

Figure 5-8 is showing the identification of the windows contained in the sixth presentation unit.

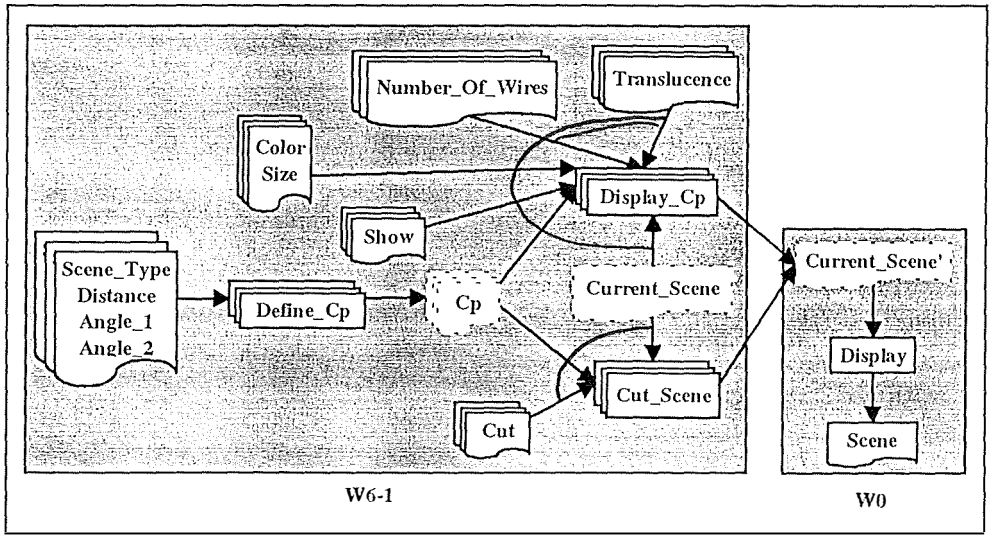


Figure 5-8 : Windows identification for the PU 6

3.7 Windows identification for the PU 7

We have identified the following logical windows concerning the PU 7 (management of the lights) :

- W7-1 → Define_lights
- W0 → Display_scene

Figure 5-9 is showing the identification of the windows contained in the seventh presentation unit.

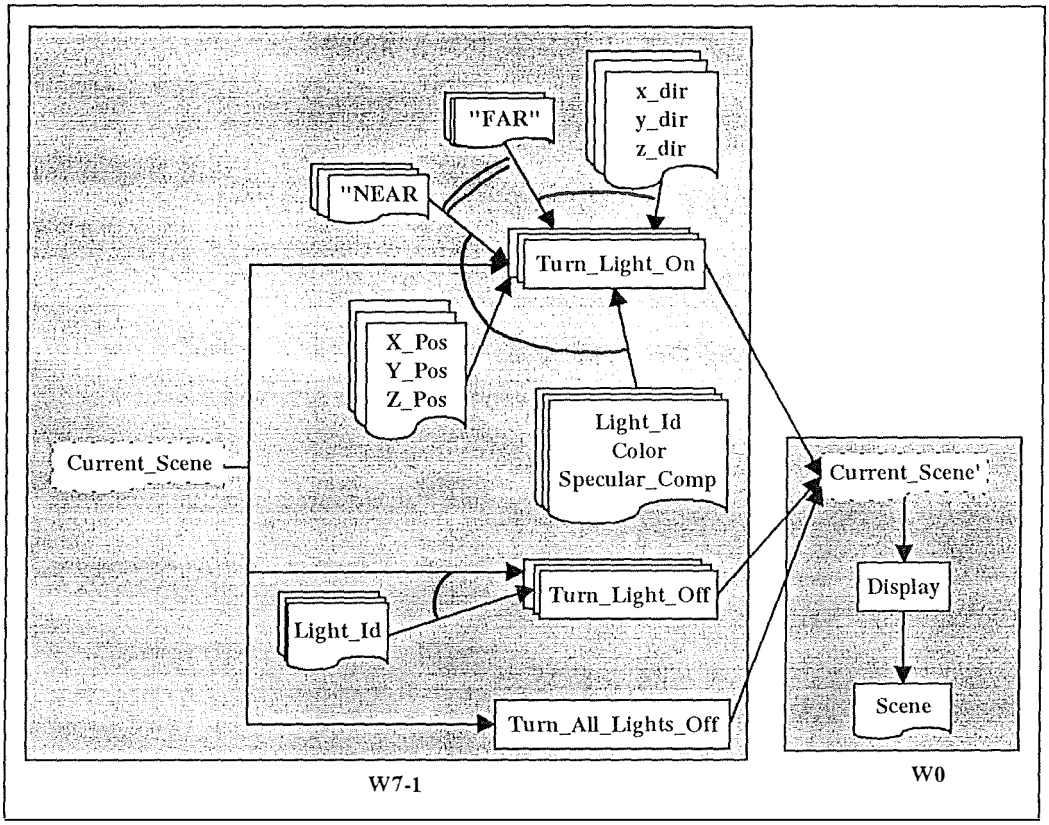


Figure 5-9 : Windows identification for the PU 7

3.8 Windows identification for the PU 8

We have identified the following logical windows concerning the PU 8 (saving into VRML format) :

- W8-1 → Save_VRML_format
- W8-2 → VRML_file_saved_message

Figure 5-10 is showing the identification of the windows contained in the eighth presentation unit. The window W8-1 is a standard one already defined in a Windows 95 library.

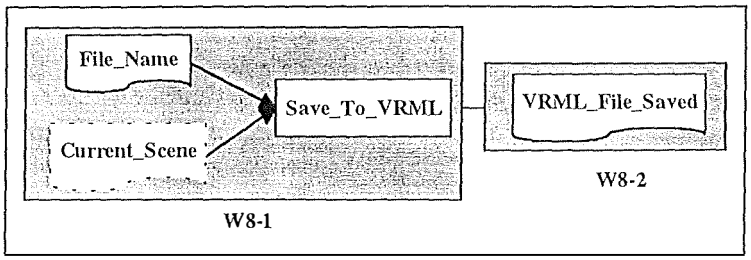


Figure 5-10 : Windows identification for the PU 8

3.9 Windows identification for the PU 9

We have identified the following logical windows concerning the PU 9 (addition of an object into the current scene) :

- W9-1 → Open_object_file_2
- W9-2 → Change_current_object_name_2
- W0 → Display_scene

Figure 5-11 is showing the identification of the windows contained in the ninth presentation unit. The window W9-1 is a standard one already defined in a Windows 95 library.

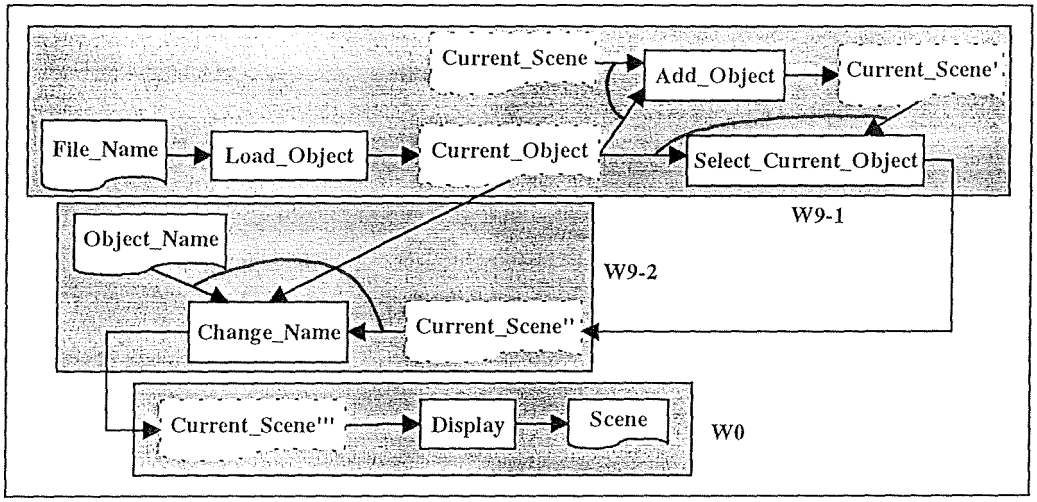


Figure 5-11 : Windows identification for the PU 9

3.10 Windows identification for the PU 10

We have identified the following logical windows concerning the PU 10 (selection of the current object) :

- W10-1 → Select_current_object

Figure 5-12 is showing the identification of the windows contained in the tenth presentation unit.

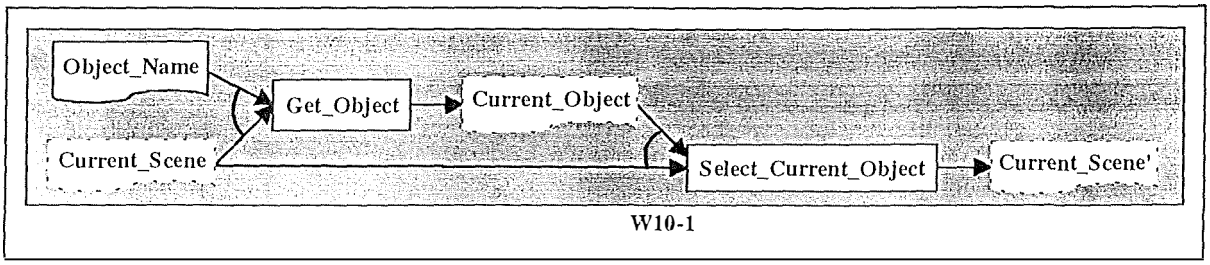


Figure 5-12 : Windows identification for the PU 10

3.11 Windows identification for the PU 11

We have identified the following logical windows concerning the PU 11 (removal of the current object from the current scene) :

- W11-1 → Remove_current_object
- W0 → Display_scene

Figure 5-13 is showing the identification of the windows contained in the eleventh presentation unit.

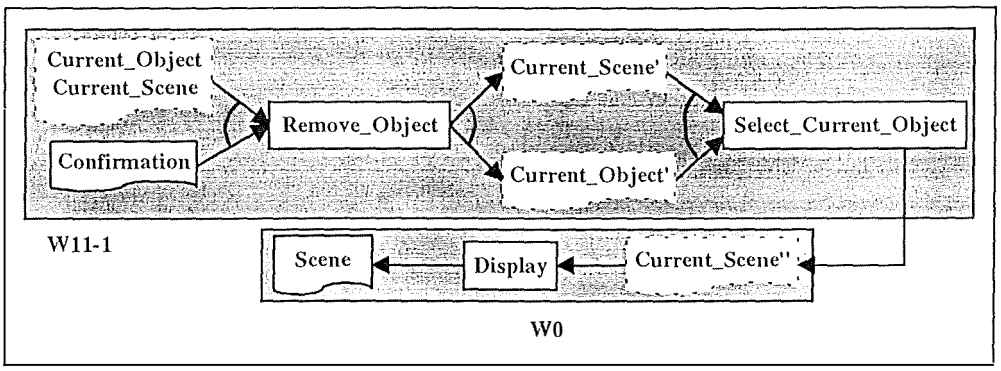


Figure 5-13 : Windows identification for the PU 11

3.12 Windows identification for the PU 12

We have identified the following logical windows concerning the PU 12 (changing the name of the current object) :

- W12-1 → Change_current_object_name_3
- W12-2 → Current_object_name_changed_message

Figure 5-14 is showing the identification of the windows contained in the twelfth presentation unit.

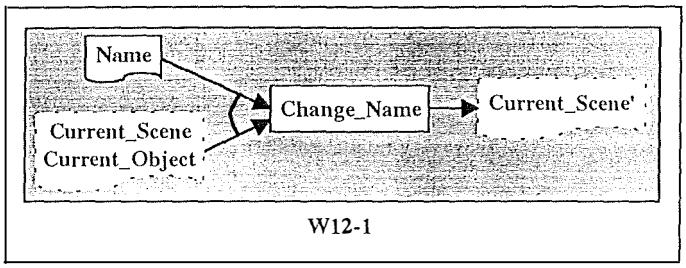


Figure 5-14 : Windows identification for the PU 12

3.13 Windows identification for the PU 13

We have identified the following logical windows concerning the PU 13 (getting information about an object) :

- W13-1 → Getting_object_information

Figure 5-15 is showing the identification of the windows contained in the thirteenth presentation unit.

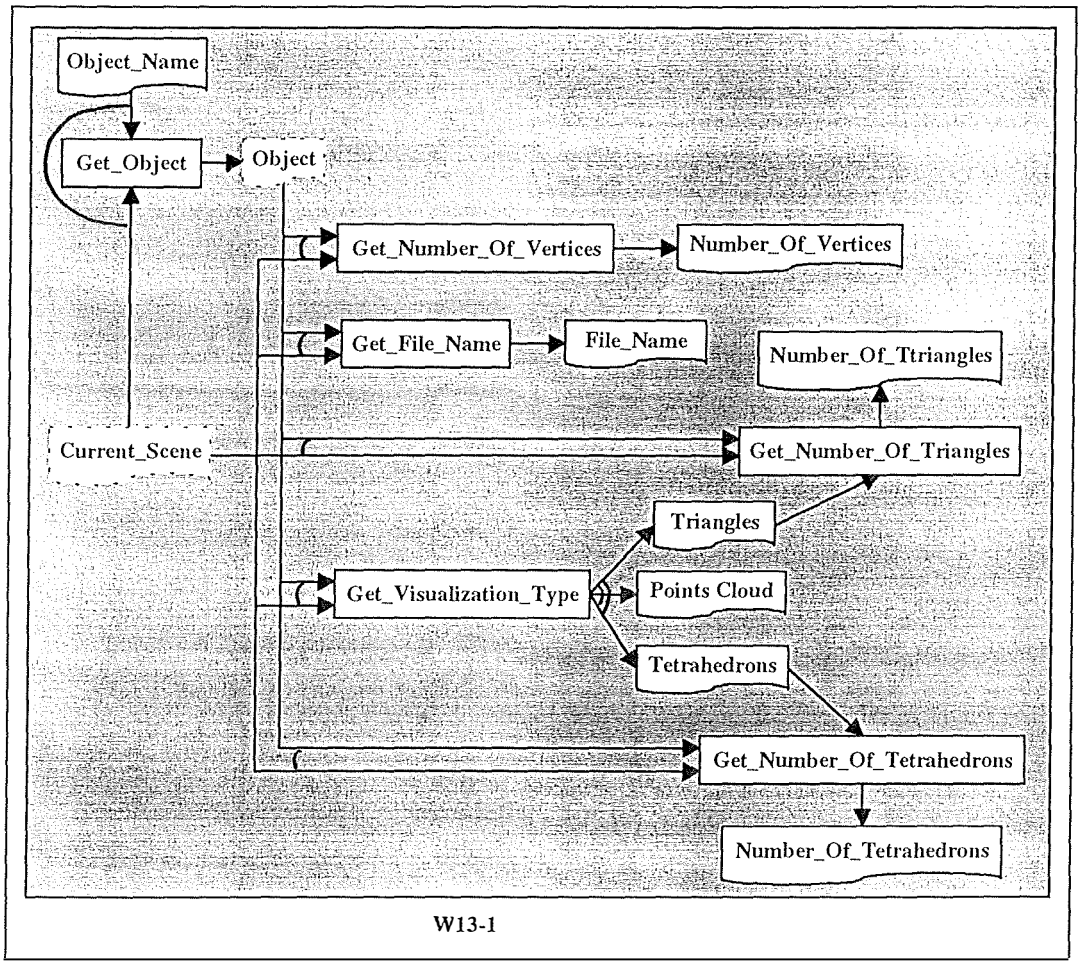


Figure 5-15 : Windows identification for the PU 13

3.14 Windows identification for the PU 14

We have identified the following logical windows concerning the PU 14 (changing the color of the current object) :

- W14-1 → Define_current_object_properties
- W0 → Display_scene

Figure 5-16 is showing the identification of the windows contained in the fourteenth presentation unit.

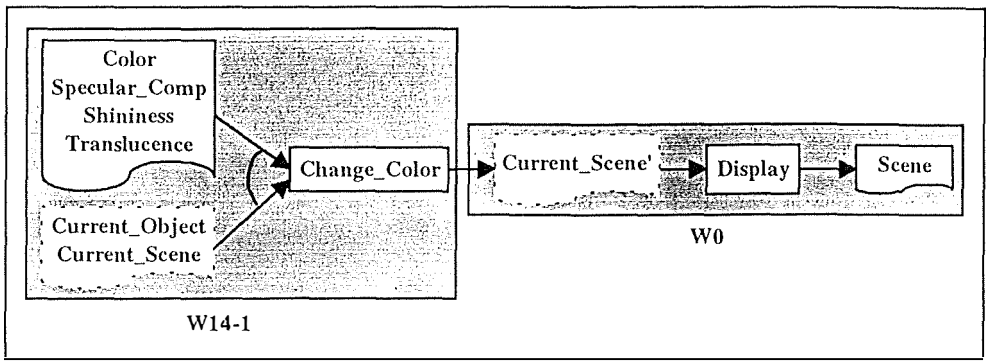


Figure 5-16 : Windows identification for the PU 14

3.15 Windows identification for the PU 15

We have identified the following logical windows concerning the PU 15 (changing the type of visualization of the current object) :

- W15-1 → Define_current_object_visualization_type
- W0 → Display_scene

Figure 5-17 is showing the identification of the windows contained in the fifteenth presentation unit.

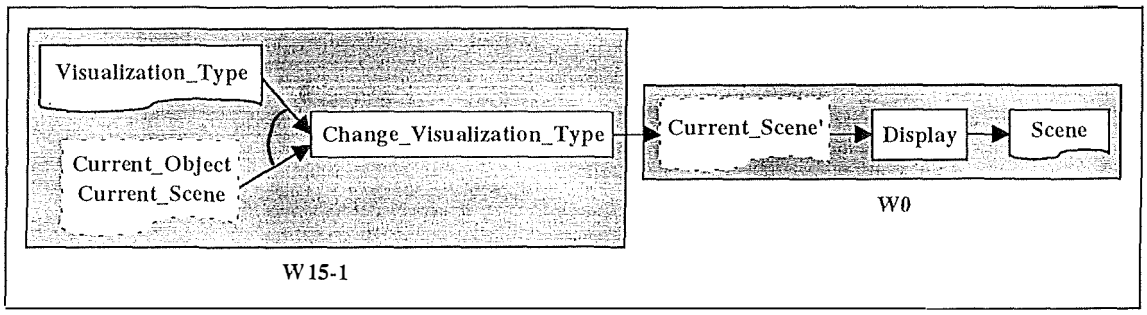


Figure 5-17 : Windows identification for the PU 15

3.16 Windows identification for the PU 16

We have identified the following logical windows concerning the PU 16 (showing the current object axis and/or box) :

- W16-1 → Define_current_object_axis_and_or_box

- W0 → Display_scene

Figure 5-18 is showing the identification of the windows contained in the sixteenth presentation unit.

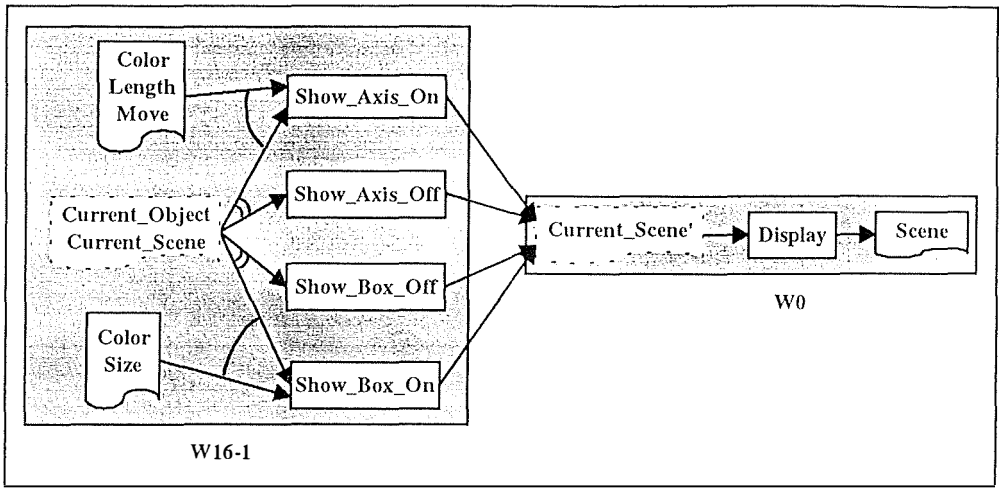


Figure 5-18 : Windows identification for the PU 16

4. AIOs selection

At this point we have to decide which types of windows (physical windows, dialog boxes or panels) will stand for the logical windows identified at the previous section. Within each window we have to select the abstract interaction objects that will correspond to each information in input/output and the one that will correspond to each function. At the end we will obtain a hierarchy of AIOs.

Except the main application window and the one that will contain the graphical representation of the scenes W0), all the other windows will be dialog boxes. Windows W1-1, W9-1 (opening an object file) and W8-1 (saving into VRML format) are dialog boxes already defined in Windows 95.

"*The selection of AIOs is based on a set of selection rules, themselves being based on empirically validated cognitive principles and established conventions*" [BODART95a]. Table 5-1, Table 5-2, Table 5-3 and Table 5-4 are showing selection rules used to choose correct AIOs respectively for alphanumeric data inputs, boolean data inputs, integer data inputs and elementary data inputs and

are coming from [VANDERDONCKT93b]. To be able to interpret these tables, here are the abbreviations used :

- Cont → continuous domain
- Exp → expandable domain
- Fr → data frequency in a list
- Lg → item current length
- Lm → maximal length of an alphanumeric item
- Nutil → user experience level
- Nvc → number of values to choose
- Npo → number of possible values
- Npv → number of principal values
- Nsv → number of secondary values
- Pref → user's preference for selecting a data
- Tm → maximal number of items in a list
- Va → antagonist values

Domain	Nvc	Nsv	Exp	Npo	Lg	AIO
Unknown					<= Lm	Single-line edit box
Unknown					> Lm	Multiple-line edit box
Mixed				[2, 3]		Radio-button with Npo items + single-line edit box
Mixed				[4, 7]		Radio-button with Npo items + single-line edit box + group box
Mixed				[8, Tm]		Drop-down combination box
Mixed				[Tm+1, 2 TM]		Scrolling combination box
Mixed				> 2 TM		Drop-down scrolling combination box
Known	> 1	= 0	No	[2, 3]		Npo check boxes
Known	> 1	= 0	No	[4, 7]		Npo check boxes + group box
Known	> 1	= 0	No	[8, Tm]		List box
Known	> 1	= 0	No	[Tm+1, 2 TM]	<= Lm	Scrolling list box
Known	> 1	= 0	No	[Tm+1, 2 TM]	> Lm	Drop-down scrolling list box
Known	> 1	= 0	No	> 2 TM		Drop-down scrolling list box
Known	> 1	= 0	Yes		<= Lm	Combination box
Known	> 1	= 0	Yes		> Lm	Drop-down combination box
Known	> 1	> 0			<= Lm	List box
Known	> 1	> 0			> Lm	Drop-down list box
Known	= 1	> 0				List box
Known	= 1	= 0	Yes			Combination box
Known	= 1	= 0	No	[2, 3]		Radio-button with Npo items

Known Known	= 1	= 0	No	[4, 7]		Radio-button with Npo items + group box
Known	= 1	= 0	No	[8, Tm]		List box
Known	= 1	= 0	No	[Tm+1, 2 TM]		Scrolling list box
Known	= 1	= 0	No	> 2 TM		Drop-down scrolling list box

Table 5-1 : AIOs selection for alphanumeric data inputs

Domain	Va	Orientation	AIO
known	Yes	Vertical	Vertical switch
known	Yes	Horizontal	Horizontal switch
known	Yes	Circular	Two-valued dial
known	Yes	Undefined	Horizontal switch
known	No		Check box
Unknown	No		Check box

Table 5-2 : AIOs selection for boolean data inputs

Nsv	Exp	Cont	Npo	Precision	Orientation	AIO
> 0						List box
= 0	Yes					Combination box
= 0	No	No	[2, 3]			Radio-button with Npo items
= 0	No	No	[4, 7]			Radio-button with Npo items + group box
= 0	No	No	[8, Tm]			List box
= 0	No	No	[Tm+1, 2 Tm]			Scrolling list box
= 0	No	No	> 2 Tm			Drop-down scrolling list box
= 0	No	Yes	[1, 10]	Low	Vertical	Scroll bar
= 0	No	Yes	[1, 10]	Low	Horizontal	Scale
= 0	No	Yes	[1, 10]	Low	Circular	Pie diagram
= 0	No	Yes	[1, 10]	Low	Undefined	Scale
= 0	No	Yes	[1, 10]	High	Vertical	Vertical thermometer
= 0	No	Yes	[1, 10]	High	Horizontal	Horizontal thermometer
= 0	No	Yes	[1, 10]	High	Circular	Dial
= 0	No	Yes	[1, 10]	High	Undefined	Horizontal thermometer
= 0	No	Yes	[11, Tm]	High		Spin button
= 0	No	Yes	[11, Tm]	Low		Scale
= 0	No	Yes	> Tm	High		Spin button
= 0	No	Yes	> Tm	Low	Vertical	Scroll bar
= 0	No	Yes	> Tm	Low	Horizontal	Scale
= 0	No	Yes	> Tm	Low	Circular	Dial
= 0	No	Yes	> Tm	Low	Undefined	Scale

Table 5-3 : AIOs selection for integer data inputs

Type	Domain	Va	Lg	Npo	Nutil	AIO
Hour					<= 5	Spin button
Hour					> 5	Profiled single-line edit box
Date					<= 5	Spin button
Date					> 5	Profiled single-line edit box
Boolean		Yes				Radio-button
Boolean		No				Check box
Graphic						Radio icon
Integer	Unknown					Single-line edit box
Integer	Mixed					Drop-down combination box
Integer	Known			[2, 7]		Radio-button
Integer	Known			[8, Tm]		Drop-down list box
Integer	Known			> Tm		Spin button
Real	Unknown					Profiled single-line edit box
Real	Mixed					Drop-down combination box
Real	Known					Drop-down list box
Alphanumeric	Unknown		<= Lm			Single-line edit box
Alphanumeric	Unknown		> Lm			Multiple-line edit box
Alphanumeric	Mixed					Drop-down combination box
Alphanumeric	Known					Drop-down list box

Table 5-4 : AIOs selection for elementary data inputs

We will now review all the windows, specify their properties and select the right AIOs they will contain taking into account the tables of selection rules presented above. For each AIO, we will determine its properties. Since software ergonomics rules are belonging to our culture, there are few differences between the AIOs we have selected during the program implementation without the help of any table and the one proposed by the tables shown above. Nevertheless, some minor modifications have to be made to the currently designed presentation, among others concerning the ergonomic rule of intra-application coherence.

4.1 Logical window W0

The logical window "Display_scene" is represented by a physical window entitled "Scene" and has the following characteristics : *modeless*, *sizable*, *minimizable*, *maximizable* and *movable*. It contains no AIOs since its only purpose is to display a view of a scene.

4.2 Logical window W1-1

The logical window "Open_object_file_1" is represented by a dialog box entitled "File Open" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and

movable. It is a standard "open file" dialog box already defined by Windows 95 and contained in one of its libraries. It should at least contain these three AIOs :

- A *single-line edit box* to enter the file name.
- An *"Ok" command button* to confirm.
- A *"Cancel" command button* to cancel the process.

4.3 Logical window W1-2

The logical window "Change_current_object_name_1" is represented by a dialog box entitled "Change object name" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *single-line edit box* to enter the name of the new object. Its default value is the string "New Object #x" where #x stands for the number of the object. For example, if the object is the fifth to be loaded into the scene, #x will be equal to 5.
- An *"Ok" command button* to accept the new name.
- A *static text* labeled "You inserted a new object. In order to continue, you must give it a name".

4.4 Logical window W2-1

The logical window "Select_current_scene" is represented only by a menu item entitled "Window". The scene can also be selected as the current one by clicking on the window containing this scene.

4.5 Logical window W3-1

The logical window "Remove_current_scene" has no external information. It's the reason why there is no window, dialog box or panel that represents this logical window. Only a click with the mouse on the cross on the upper right corner or a click on a menu item on the upper left of the window containing the current scene will destroy this window.

4.6 Logical window W4-1

The logical window "Choose_scene_parameters" is represented by a dialog box entitled "Scene options" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *spin button* to choose the near-plane (an integer between 1 and 20).
- A *spin button* to choose View angle or field of view (an integer between 10 and 120).
- A *group box* entitled "Size" to gather the previous two spin buttons because they share the same semantic.

- A *radio button* to choose the flat shading method.
- A *radio button* to choose the smooth shading method.
- A *group box* entitled "Shading Method" to gather the previous two radio buttons because they share the same semantic.
- A *radio button* to choose the clockwise vertices sorting method.
- A *radio button* to choose the counter-clockwise vertices sorting method.
- A *group box* entitled "Vertices Sorting Method" to gather the previous two radio buttons because they share the same semantic.
- A *"Change..." command button* to call the window that permits to change the background color of the scene.
- An *icon* to show the current backgroud color.
- A *group box* entitled "Background Color" to gather the previous two AIOs because they share the same semantic.
- A *check box* to select the antialiasing option or not.
- A *check box* to select the culling face option or not.
- A *group box* entitled "Other" because they can not be placed anywhere else and in order to keep symmetry in the dialog box presentation.
- An *"Ok" command button* to accept the new parameters of the scene.
- A *"Cancel" Command button* to keep the old parameters of the scene.

4.7 Logical window W5-1

The logical window "Rotate_scene" is represented by a dialog box entitled "Set angles" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *spin button* to choose the value of the new angle of the objects along the x axis.
- A *spin button* to choose the value of the new angle of the objects along the y axis.
- A *spin button* to choose the value of the new angle of the objects along the z axis.
- A *group box* entitled "New Angles" to gather the previous three spin buttons because they share the same semantic.
- A *single-line edit box* to show the value of the curent angle of the objects along the x axis. The user can't modify this edit box.
- A *single-line edit box* to show the value of the curent angle of the objects along the y axis. The user can't modify this edit box.
- A *single-line edit box* to show the value of the curent angle of the objects along the z axis. The user can't modify this edit box.
- A *group box* entitled "Current Angles" to gather the previous three single-line edit boxes because they share the same semantic.

- An *"Ok" command button* to accept the parameters and to perform the rotation.
- A "*Cancel" Command button* not to perform the rotation and keep the angles of all the objects at their current values.
- An *"Initial view" command button* to set the objects of the scene at their very first rotation angle.

4.8 Logical window W5-2

The logical window "Translate_scene" is represented by a dialog box entitled "Translate" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *spin button* to choose the value of the new position of the objects along the x axis.
- A *spin button* to choose the value of the new position of the objects along the y axis.
- A *spin button* to choose the value of the new position of the objects along the z axis.
- A *group box* entitled "New Position" to gather the previous three spin buttons because they share the same semantic.
- A *single-line edit box* to show the value of the curent position of the objects along the x axis. The user can't modify this edit box.
- A *single-line edit box* to show the value of the curent position of the objects along the y axis. The user can't modify this edit box.
- A *single-line edit box* to show the value of the curent position of the objects along the z axis. The user can't modify this edit box.
- A *group box* entitled "Current Positions" to gather the previous three single-line edit boxes because they share the same semantic.
- A *radio button* to choose the best fit translation method.
- A *radio button* to choose the absolute translation method.
- A *radio button* to choose the relative translation method.
- A *group box* entitled "Method" to gather the previous three radio buttons because they share the same semantic.
- An *"Ok" command button* to accept the parameters and to perform the translation.
- A "*Cancel" Command button* not to perform the translation and keep the objects at their current position values.
- A *"Center" command button* to center all the objects in the middle of the scene.

4.9 Logical window W5-3

The logical window "Scale_scene" is represented by a dialog box entitled "Set Scale" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *spin button* to choose the value of the new scale percentage of the objects along the x axis.
- A *spin button* to choose the value of the new scale percentage of the objects along the y axis.
- A *spin button* to choose the value of the new scale percentage of the objects along the z axis.
- A *check box* to maintain the global aspect ratio or not.
- A *group box* entitled "Change aspect ratio" to gather the previous four AIOs because they share the same semantic.
- A *single-line edit box* to show the value of the curent scale percentage of the objects along the x axis. The user can't modify this edit box.
- A *single-line edit box* to show the value of the curent scale percentage of the objects along the y axis. The user can't modify this edit box.
- A *single-line edit box* to show the value of the curent scale percentage of the objects along the z axis. The user can't modify this edit box.
- A *group box* entitled "Current Aspect Ratio" to gather the previous three single-line edit boxes because they share the same semantic.
- An *"Ok" command button* to accept the parameters and to perform the scaling.
- A *"Cancel" Command button* not to perform the scaling and keep the objects at their current scale percentage values.

4.10 Logical window W6-1

The logical window "Define_cutting_planes" is represented by a box entitled "Cutting Planes" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *list box* to choose which cutting plane to set the parameters. It should contain the following alphanumeric elements : "Bottom plane", "Front Plane", "Left plane", "Rear plane", "Right plane", and "Top plane".
- A *check box* to show the cutting plane or not.
- A *check box* to cut the scene or not.
- A *spin button* to choose the value of the distance percentage between the cutting plane and the limit plane of the scene.
- A *spin button* to choose the value of the angle between the cutting plane and the limit plane of the scene along the x axis.
- A *spin button* to choose the value of the angle between the cutting plane and the limit plane of the scene along the y axis.
- A *spin button* to choose the value of the angle between the cutting plane and the limit plane of the scene along the z axis.
- A *group box* entitled "Planes Parameters" to gather the previous seven AIOs because they share the same semantic.

- A *"Pick..." command button* to call the window that permits to change the color of the cutting plane.
- An *icon* to show the current cutting plane color.
- A *group box* entitled "Other" to gather the previous two AIOs because they share the same semantic.
- An *"Ok" command button* to accept the parameters and to perform the cutting planes.
- A *"Cancel" Command button* to keep the scene as it was before (no cutting plane has changed).
- A *"Default" Command button* to set the parameters of all the cutting planes with default values (no active cutting plane, no visible cutting planes, default color...).
- An *"Advanced >>" Command button* to show the same window but bigger and with more AIOs on it.

This dialog box is expandable. When the "Advanced >>" Command button is pushed by the user, the same dialog box becomes bigger and contains more AIOs concerning advanced options. We will consider that it is another dialog box than the previous one that appear on the screen. The first "Cutting Plane" dialog box that is already described will be identified by W6-1a when the second window, the one that will be described below, will be identified by W6-1b.

Dialog box W6-1b has the same characteristics and the same AIOs as dialog box W6-1a except that the "Advanced >>" Command button of the latter window is replaced by :

- An *"<< Advanced" Command button* to show the same window but smaller and with less AIOs on it.
- Moreover, dialog box W6-1b has the following AIOs that dialog box W6-1a has not :
- A *radio button* to choose the wireframe visual aspect of the cutting plane.
- A *spin button* to choose the number of wires the plane should have (value between 0 and 100).
- A *radio button* to choose the translucent visual aspect of the cutting plane.
- A *spin button* to choose the translucence percentage the plane should have.
- A *spin button* to choose the size of the plane (a percentage of the scene size).
- A *group box* entitled "Advanced" to gather the previous five AIOs because they share the same semantic.

4.11 Logical window W7-1

The logical window "Define_lights" is represented by a dialog box entitled "Lighting" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *list box* to choose which light to set the parameters of. It should contain the folowing alphanumeric elements : "Light 1", ..., "Light 2".

- A *check box* to turn the light on or off.
- A *group box* entitled "Available Lights" to gather the previous two AIOs because they share the same semantic.
- A *"Pick..." command button* to call the window that permits to change the color of the light.
- An *icon* to show the light color.
- A *spin button* to choose specular component of the light (value between 0 and 255).
- An *icon* to show the light specular component.
- A *group box* entitled "Light Components" to gather the previous four AIOs because they share the same semantic.
- A *single-line edit box* to enter the light position or the light direction along the x axis.
- A *single-line edit box* to enter the light position or the light direction along the y axis.
- A *single-line edit box* to enter the light position or the light direction along the z axis.
- A *check box* to decide if the light is far or near to the objects.
- A *group box* entitled "Light Position" to gather the previous four AIOs because they share the same semantic.
- A *check box* to durn the all the light off or to enable the lighting effects.
- An *"Ok" command button* to accept the parameters and to perform the lighting.
- A *"Cancel" Command button* to keep the scene as it was before (no light has changed).
- A *"Default" Command button* to parametrize all the lights with default values (no active light, default color...).

4.12 Logical window W8-1

The logical window "Save_VRML_format" is represented by a dialog box entitled "Save File" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It is a standard "save file" dialog box already defined by Windows 95 and contained in one of its libraries. It should at least contain these three AIOs :

- A *single-line edit box* to enter the file name.
- An *"Ok" command button* to save the file.
- A *"Cancel" command button* to cancel the process.

4.13 Logical window W8-2

The logical window "VRML_file_saved_message" is represented by a dialog box entitled "Exporting in VRML" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIO :

- A *Progression indicator* to show the current state of the file saving.

4.14 Logical window W9-1

The logical window "Open_object_file_2" is represented by a dialog box entitled "File Open" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It is a standard "open file" dialog box already defined by Windows 95 and contained in one of its libraries. It should at least contain these three AIOs :

- A *single-line edit box* to enter the file name.
- An *"Ok" command button* to confirm.
- A *"Cancel" command button* to cancel the process.

4.15 Logical window W9-2

The logical window "Change_current_object_name_2" is represented by a dialog box entitled "Change object name" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *single-line edit box* to enter the name of the new object. Its default value is the string "New Object #x" where #x stands for the number of the object. For example, if the object is the fifth to be loaded into the scene, #x will be equal to 5.
- An *"Ok" command button* to accept the new name.
- A *static text* labeled "You inserted a new object. In order to continue, you must give it a name".

4.16 Logical window W10-1

The logical window "Select_current_object" is represented by a dialog box entitled "Select an object" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *list box* to choose which object to select as the current one. The list box contains all the names of the objects present in the scene.
- A *group box* entitled "Objects" that enclose the previous list box to give it a title.
- An *"Ok" command button* to accept the new current object.
- A "*Cancel" Command button* to keep the old current object.

4.17 Logical window W11-1

The logical window "Remove_current_object" is represented by a dialog box entitled "Remove current object" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *static text* labeled "Are you sure you want to remove object the current object ?".

- A *"Yes" command button* to process the deletion of the current object.
- A *"No" command button* to cancel the deletion process.

4.18 Logical window W12-1

The logical window "Change_current_object_name_3" is represented by a dialog box entitled "Change Object Name" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *single-line edit box* to enter the new name of the current object.
- An *"Ok" command button* to accept the new name.
- A *"Cancel" command button* to keep the old name.

4.19 Logical window W13-1

The logical window "Getting_object_information" is represented by a dialog box entitled "Objects infos" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *list box* to choose the object we need information for. The list box contains all the names of the objects present in the scene.
- A *group box* entitled "Select an object" that encloses the previous list box to give it a title.
- A *single-line edit box* to show the value of the number of elements (tetrahedrons or triangles) of the object. The user can't modify this edit box.
- A *single-line edit box* to show the value of the number of vertices of the object. The user can't modify this edit box.
- A *single-line edit box* to show the display type (mesh, filled surface or points cloud) of the object. The user can't modify this edit box.
- A *single-line edit box* to show the file name of the object. The user can't modify this edit box.
- A *group box* entitled "Mesh" to gather the previous four single-line edit box because they share the same semantic.
- A *"Close" command button* to close the window when the user has read the information he needs.

4.20 Logical window W14-1

The logical window "Define_current_object_properties" is represented by a dialog box entitled "Object Properties" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *check box* to decide if the object surface is transparent or not.
- A *spin button* to choose the transparency percentage.

- A *group box* entitled "Transparency" to gather the previous two AIOs because they share the same semantic.
- A *"Pick..." command button* to call the window that permits to change the color of the object.
- An *icon* to show the current object color.
- A *spin button* to choose the specular component (gray levels between 0 and 255).
- A *spin button* to choose the shininess percentage.
- A *group box* entitled "Colors" to gather the previous four AIOs because they share the same semantic.
- An *"Ok" command button* to accept the new properties of the object.
- A *"Cancel" command button* to keep the old properties of the object.
- A *"Default" Command button* to set the parameters of all the object properties with default values (no transparent surface, default color...).

4.21 Logical window W15-1

The logical window "Define_current_object_visualization_type" is represented by a dialog box entitled "Display" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *radio button* to choose the filled surface (triangles) display type.
- A *radio button* to choose the mesh (tetrahedrons) display type.
- A *radio button* to choose the points cloud display type.
- A *group box* entitled "Surface display" to gather the previous three radio buttons because they share the same semantic.
- An *"Ok" command button* to accept the new visualization type.
- A *"Cancel" command button* to keep the old visualization type.
- A *"Default" Command button* to choose the default visualization type.

4.22 Logical window W16-1

The logical window "Define_current_object_axis_and_or_box" is represented by a dialog box entitled "Axis and box" and has the following characteristics : *modal*, *non-sizable*, *non-minimizable*, *non-maximizable* and *movable*. It contains the following AIOs :

- A *check box* to decide if an axis has to be shown or not with the current object.
- A *check box* to decide if the axis has to move or not with the current object.
- A *"Pick..." command button* to call the window that permits to change the color of the axis.
- An *icon* to show the color of the current object axis.
- A *spin button* to choose the axis length percentage (percentage of the object size).

- A *group box* entitled "Axis Properties" to gather the previous five AIOs because they share the same semantic.
- A *check box* to decide if a box has to be shown or not with the current object.
- A *"Pick..." command button* to call the window that permits to change the color of the box.
- An *icon* to show the color of the current object box.
- A *spin button* to choose the box size percentage (percentage of the object size).
- A *group box* entitled "Box Properties" to gather the previous four AIOs because they share the same semantic.

5. Transformation of the AIOs into CIOs

Each AIO identified in the previous section will be connected to a concrete interactive object coming from the Microsoft Windows 95 environment. All the CIOs obtained in this part will be used to physically design the dialog boxes. The conversion table from AIOs to CIOs is shown in Table 5-5.

AIOs	CIOs
Spin button	Spin button
Icon	Icon
Command button	Push button
Check box	Check box
Group box	Group box
Radio button	Radio button
Single-line edit box	Single line entry field
List box	List box
Static text	Static text
Progression indicator	Progression indicator
Drop-down list box	Drop-down list box
Modeless dialog box	Modeless dialog box
Modal dialog box	Modal dialog box

Table 5-5 : Transformation of the AIOs into CIOs

6. CIOs placement and manual edition of the presentation

In this section we will show all the windows and dialog boxes that are currently implemented in our application. The CIOs placement was manually done. A word about the placement strategy :

- The mnemonic terms are unique and displayed if available.
- The CIOs are arranged in an logical and aesthetic ways.
- Except for the dialog boxes where only output information is displayed, all the other windows must have an "Ok" push button and a "Cancel" push button.
- If possible, windows should have a "Default" push button to set default values to input CIOs.
- The push buttons are, if possible, placed horizontally, on the right side of the dialog box.
- Every CIO must have a label.
- Each CIO concerning the color or its specular component must be represented by a push button calling a standard Windows 95 window that help the user to choose the color and by an icon showing the current color used.

These are the screenshots of the windows designed with Microsoft Visual C++. The windows shown below were initially created during our training period in Tony Keller's lab but they were modified taking into account the presentation design suggested by the TRIDENT methodology.

6.1 Window W1-2 : Change_Current_Object_Name_1

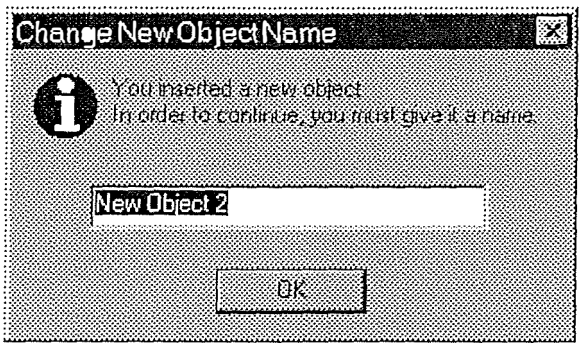


Figure 5-19 : Window W1-2

The window shown in Figure 5-19 is displayed when an object is opened. It allows to give the object a name different from a single number. For example, one object can be "Surface mesh" and another one in the same scene "Volume mesh".

6.2 Window W4-1 : Choose_Scene_parameters

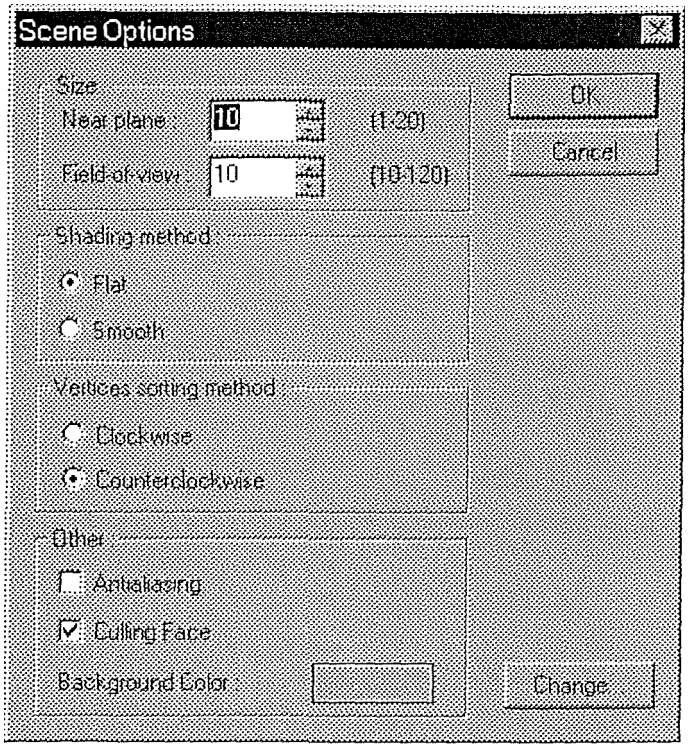


Figure 5-20 : Window W4-1

The window shown in Figure 5-20 allows to set parameters for the scene, as the position of far and near planes.

6.3 Window W5-1 : Rotate_Scene

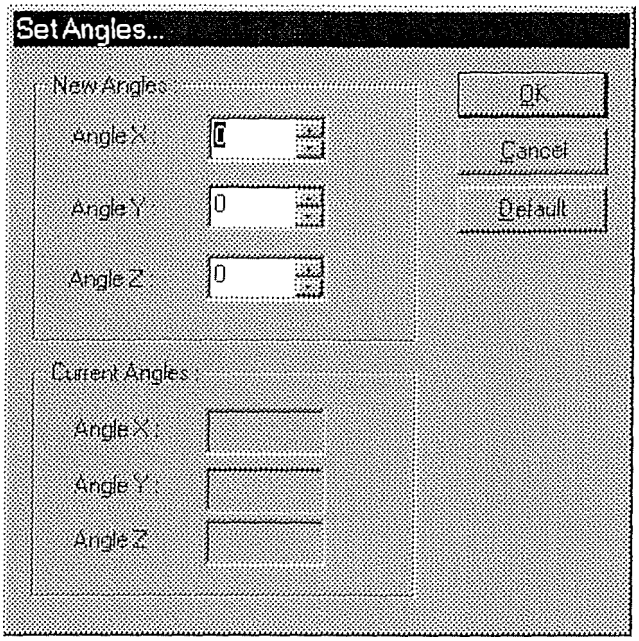


Figure 5-21 : Window W5-1

The window shown in Figure 5-21 allows the user to rotate the scene. The new angle is set in the first part of the window, the second one shows the current position from the axis.

6.4 Window W5-2 : Translate_Scene

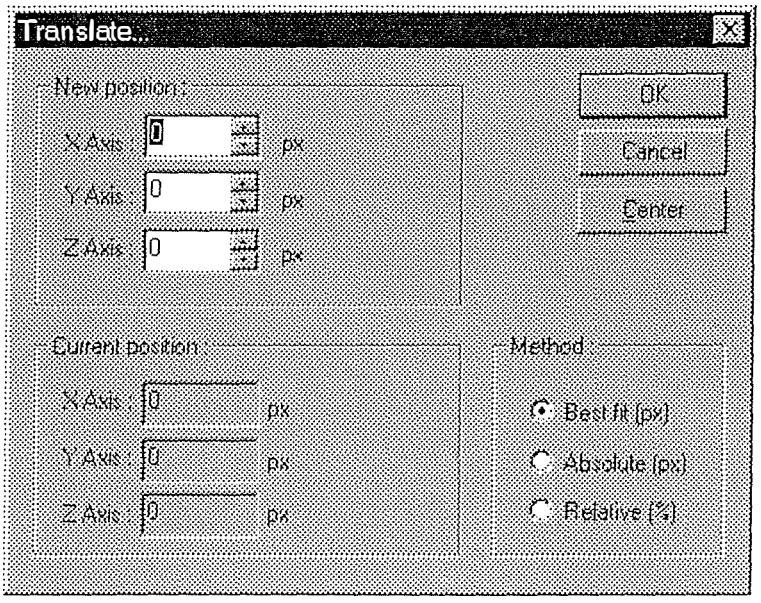


Figure 5-22 : Window W5-2

As for W5-1, this window allows to change the position of the object. The current position is shown in the second part of the dialog box. The bottom right group box corresponds to the method that can be used. (Figure 5-22)

6.5 Window W5-3 : Scale Scene

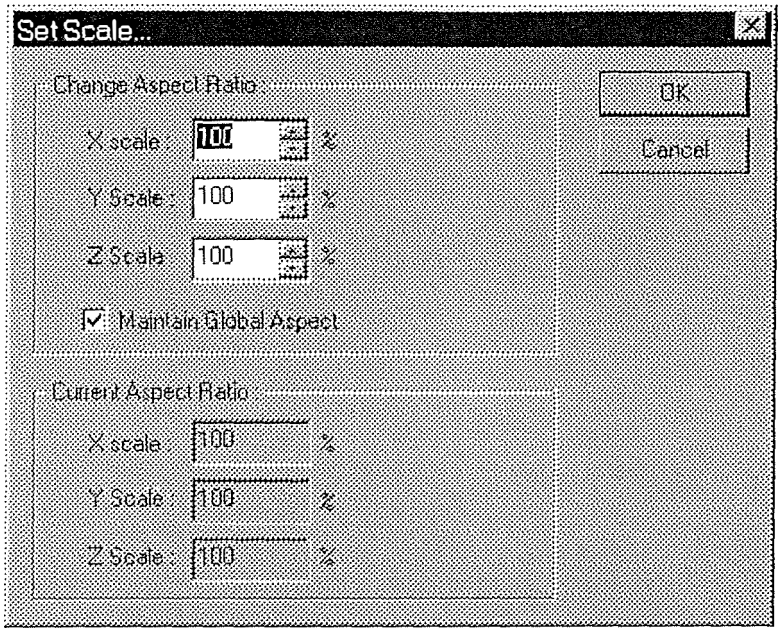


Figure 5-23 : Window W5-3

The window shown in Figure 5-23 allows to change the aspect of the objects. While "Maintain Global Aspect" button is checked, X, Y and Z scales are set together, otherwise, it is possible to change aspect ratio for X, Y and Z axes separately.

6.6 Window W6-1 : Define_Cutting_Planes

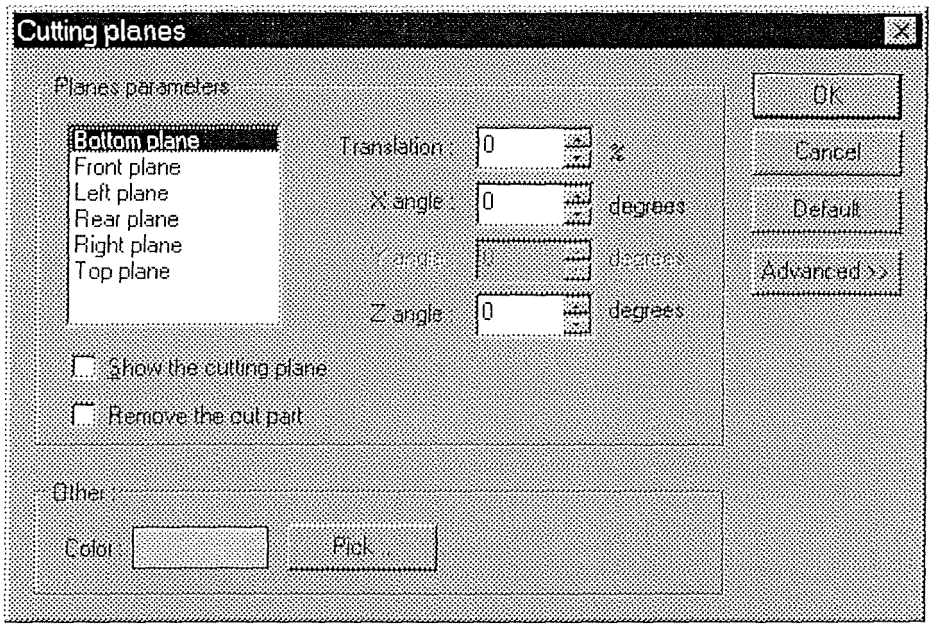


Figure 5-24 : Window W6-1a (standard window)

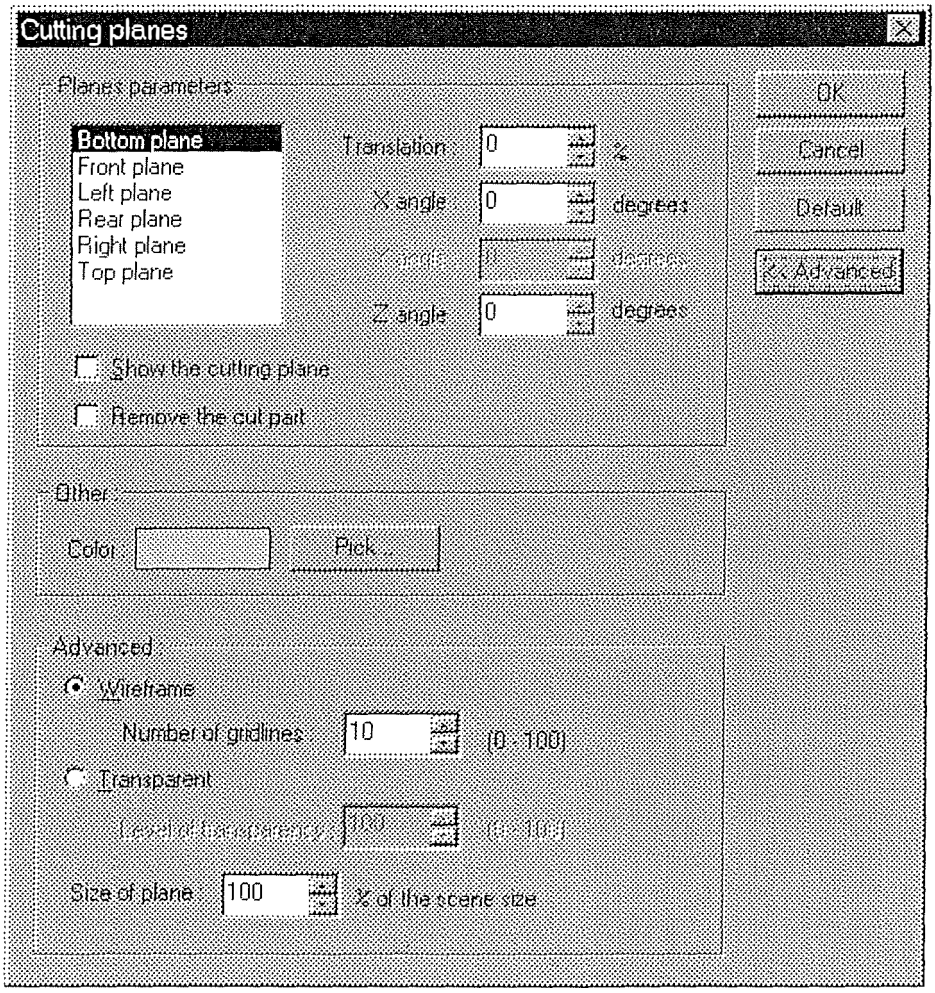


Figure 5-25 : Window W6-1b (Advanced options)

The two windows shown in Figure 5-24a and Figure 5-25 correspond to the dialog box allowing to change parameters for cutting planes (see Chapter Visualization). There are actually two windows, one for common parameters, and a second one with advanced parameters.

6.7 Window W7-1 : Define_Lights

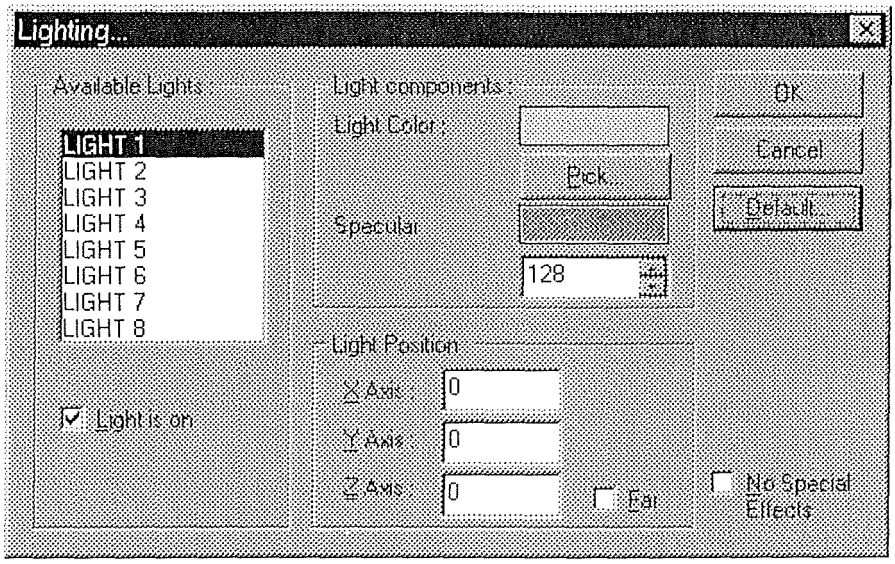


Figure 5-26 : Window W7-1

The window shown in Figure 5-26 allows to set parameters for lights. Up to 8 lights can be set simultaneously, and for each, the color, specular component and position can be set.

6.8 Window W8-1 : Save_VRML_Format

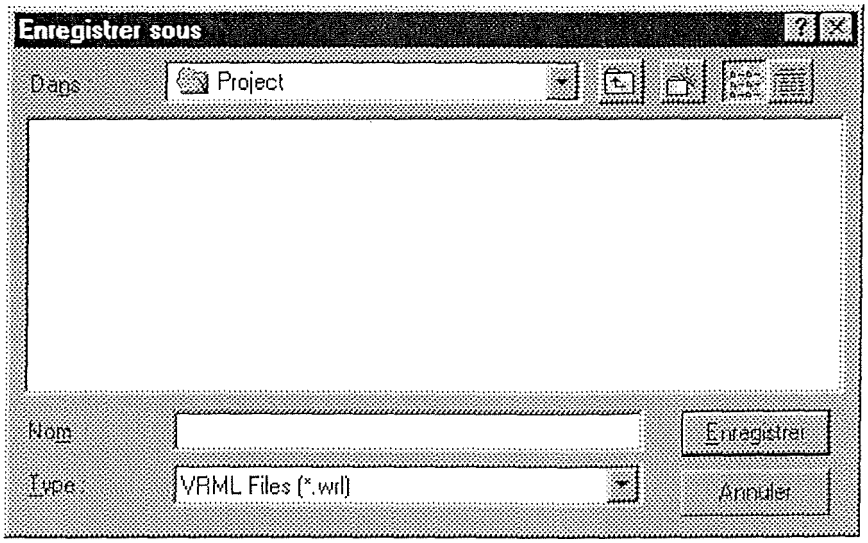


Figure 5-27 : Window W8-1

The window shown in Figure 5-27 is the common dialog "Save As"[30] where the type of file is "VRML Files" (See Appendix 5 for details)

6.9 Window W8-2 : VRML_file_saved_message

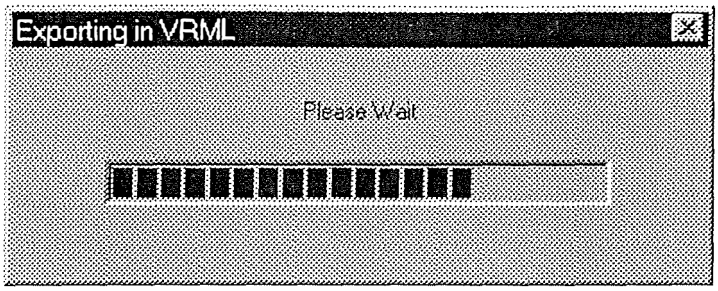


Figure 5-28 : Window W8-2

The window shown in Figure 5-28 is displayed when saving the VRML file.

6.10 Window W9-1 : Open_Object_File_2

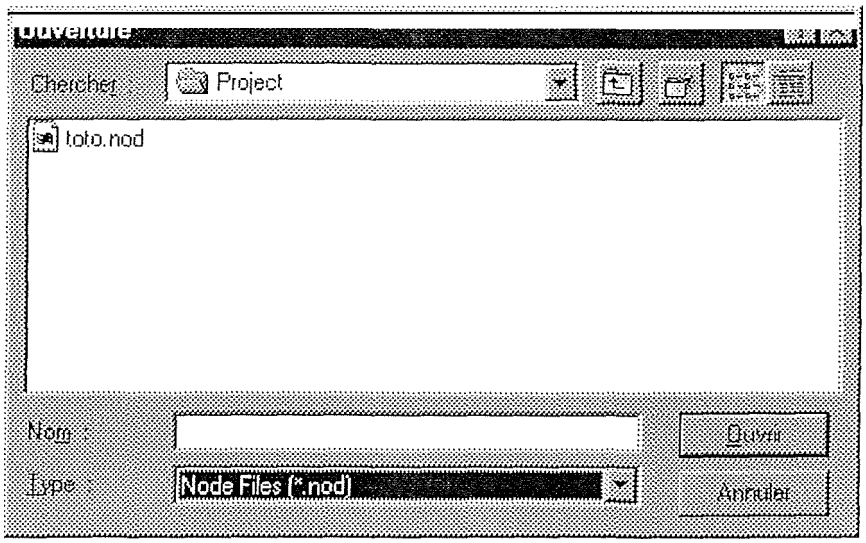


Figure 5-29 : Window W9-1

The window W9-1 as shown in Figure 5-29 corresponds to the common dialog "Open File" dialog window.

[30] This dialog box as well as "Open File" as Common Dialogs provided with Windows 95 and Windows NT. Unfortunately, screen captures were made under the French release of Windows 95.

6.11 Window W9-2 : Change_Current_Object_Name_2

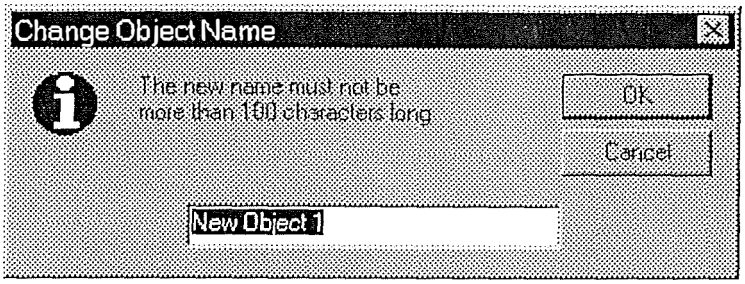


Figure 5-30 : Window W9-2

The window shown in Figure 5-30 is displayed when a new object is inserted in the scene.

6.12 Window W10-1 : Select_Current_Object

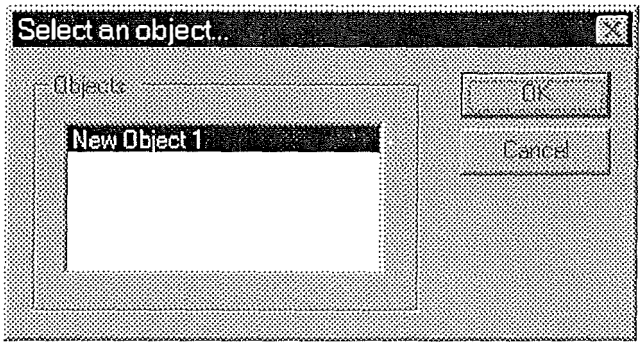


Figure 5-31 : Window W10-1

Figure 5-31 shows the window allowing to select an object from the scene and to make it the current object.

6.13 Window W11-1 : Remove_Current_Object

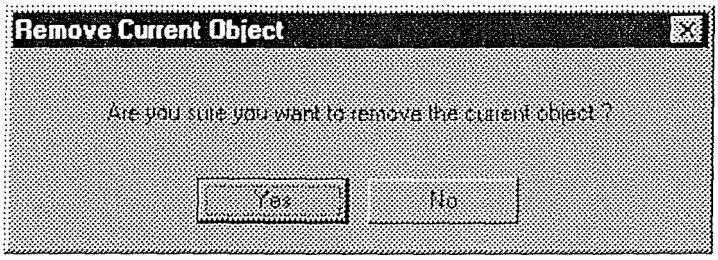


Figure 5-32 : Window W11-1

The window shown in Figure 5-32 is displayed when the menu item "Remove Current Object" is activated.

6.14 Window W12-1 : Change_Current_Object_Name_3

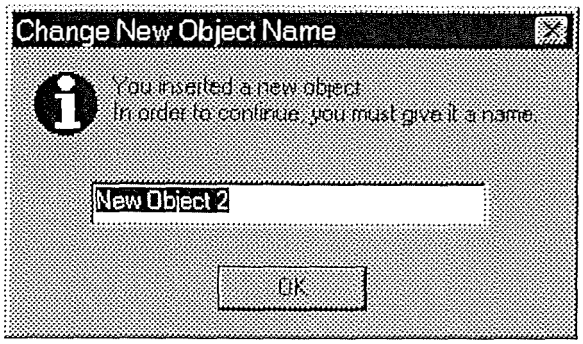


Figure 5-33 : Window W12-1

The window shown in Figure 5-33 is displayed when the menu item "Rename Current Object" is activated.

6.15 Window W13-1 : Getting_Object_Information

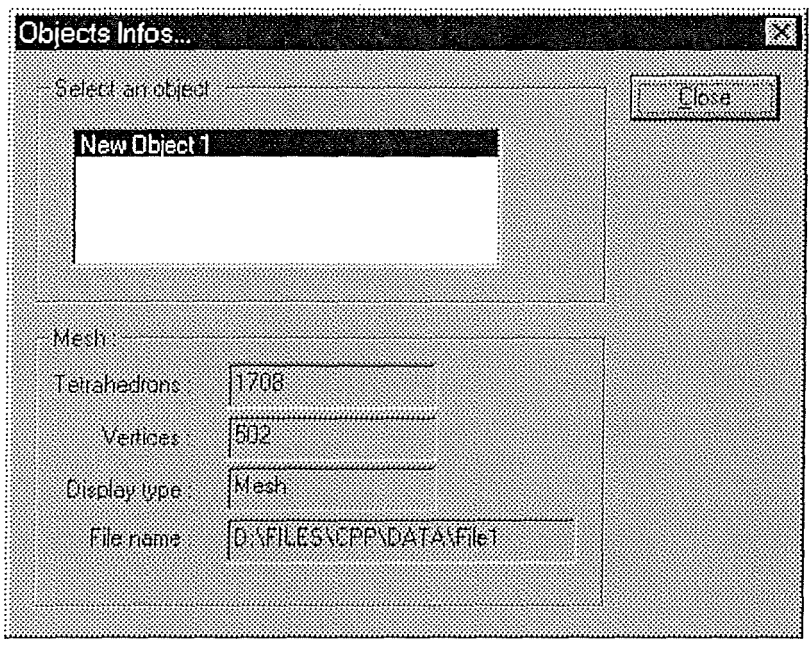


Figure 5-34 : Window W13-1

The window shown in Figure 5-34 displays all objects from the scene. Once an object is selected in the list box, information is displayed.

6.16 Window W14-1 : Define_Current_Object_Properties

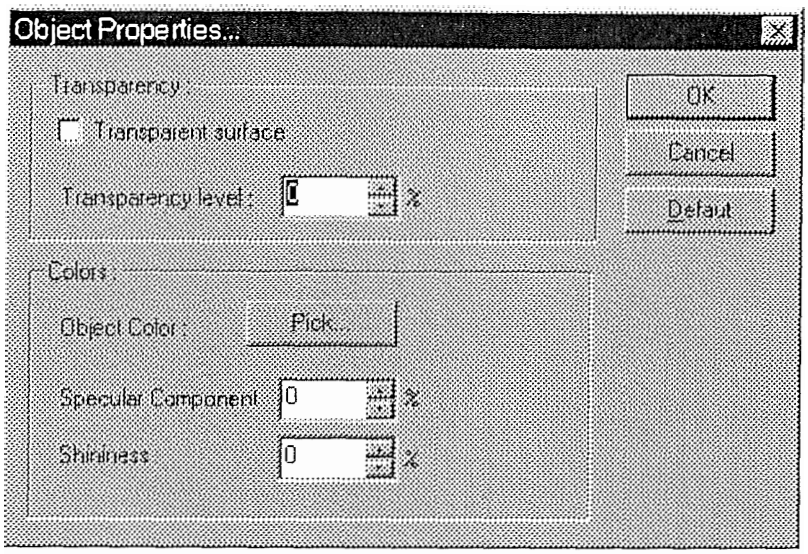


Figure 5-35 : Window W14-1

Figure 5-35 shows the window that allows to set properties like transparency and color.

6.17 Window W15-1 : Define_Current_Object_Type

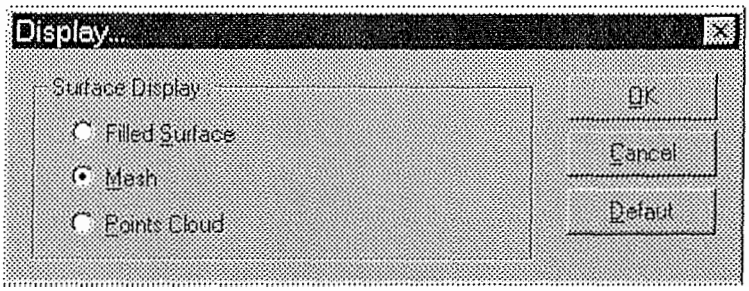


Figure 5-36 : Window W15-1

The window W15-1 shown in Figure 5-36 allows to change the visualization type (rendered surface, mesh or points cloud).

6.18 Window W16-1 : Define_Current_Object_Axis_and_or_Box

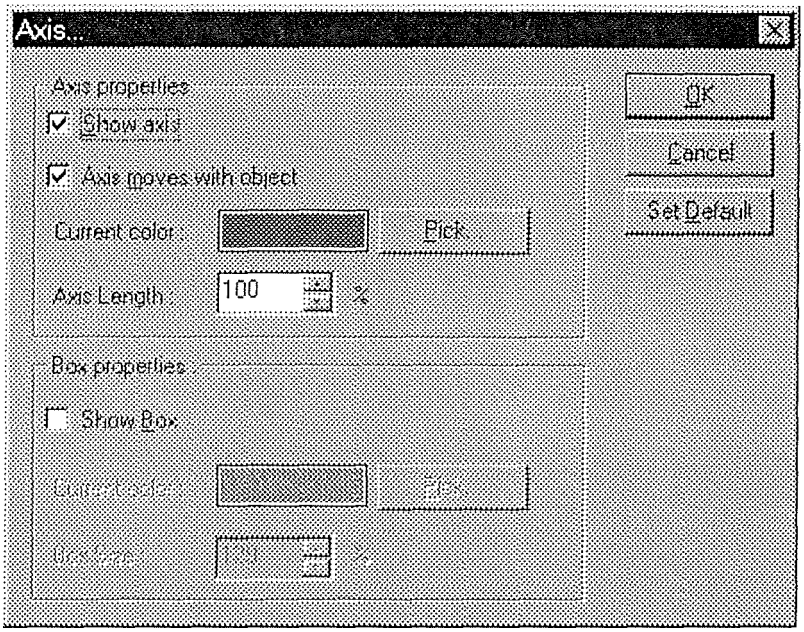


Figure 5-37 : Window W16-1

The window shown in Figure 5-37 allows to display or not axis and/or bounding box.

7. Using an Expert System for Automatic Generation of User Interface.

We used below a program called SEGUIA [SEGUIA97] to validate our dialog boxes. The process has not been conducted for all of them since most of them are simple and the result would be the same. We have chosen the "Cutting planes" dialog box because it offers a lot of different control interactive objects (CIOs). The dialog box shown in Figure 5-38 was manually designed under Microsoft Developer Studio (The software that allows to develop C++ code as well as resources like dialog boxes) and was included in 3D Viewer. Figure 5-39 shows a dialog box created by SEGUIA. The same parameters as in the first dialog box were given to SEGUIA, and in many respects, they are similar.

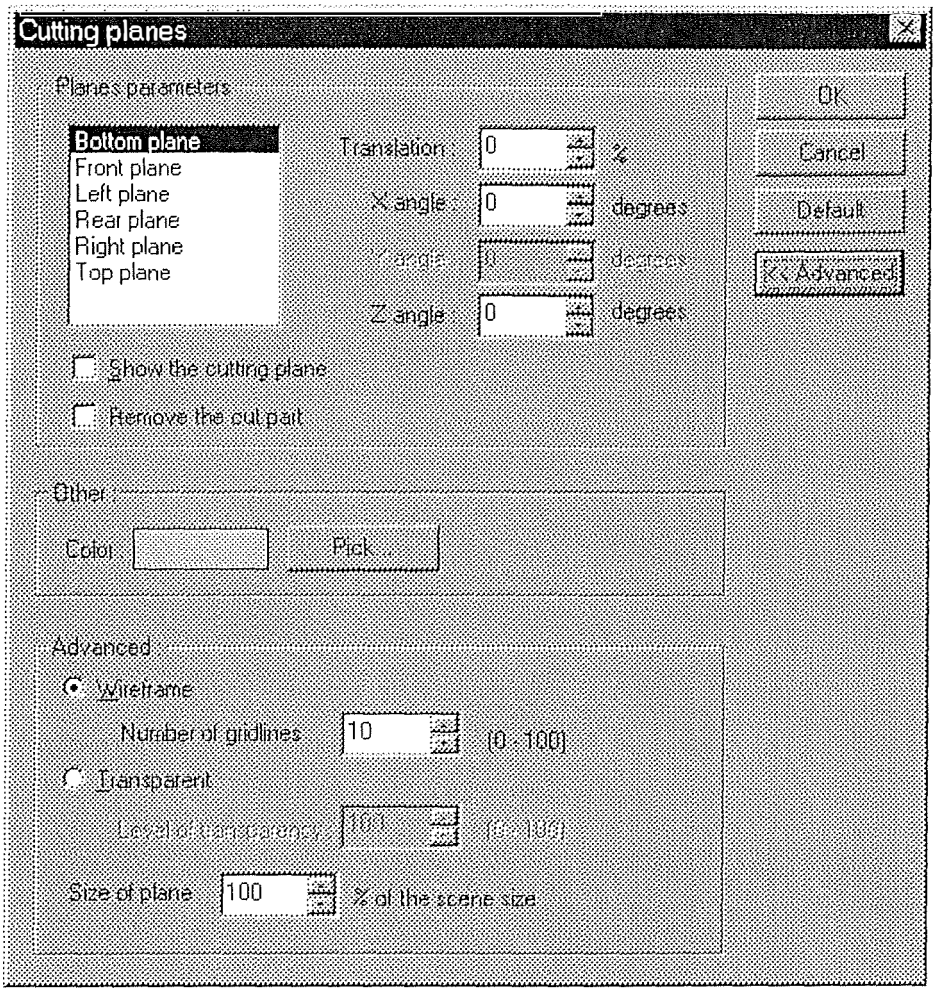


Figure 5-38 : Dialog box excerpt from 3D Viewer

Indeed, parameters like translation, angle and number of gridlines are represented by the same CIOs in both dialog boxes (spin button) because the range of acceptable values is wide. However, several differences appear. For example, available planes are not represented the same way. We list them into a list box but SEGUIA uses radio buttons. The use of radio buttons matches with ergonomic rules that state that when the domain is known, and the number of possible values range from 4 to 7, the radio buttons are to be used. [VANDERDONCKT93b].

These radio buttons could have been placed in a more significant way, as shown in Figure 5-40, but on the other hand it overloads the interface. The grouping of objects is also slightly different. Since the number of elements is not to high, only one dialog box was created where we decided to create a small one with only basic parameters and an other one with basic and advanced parameters.

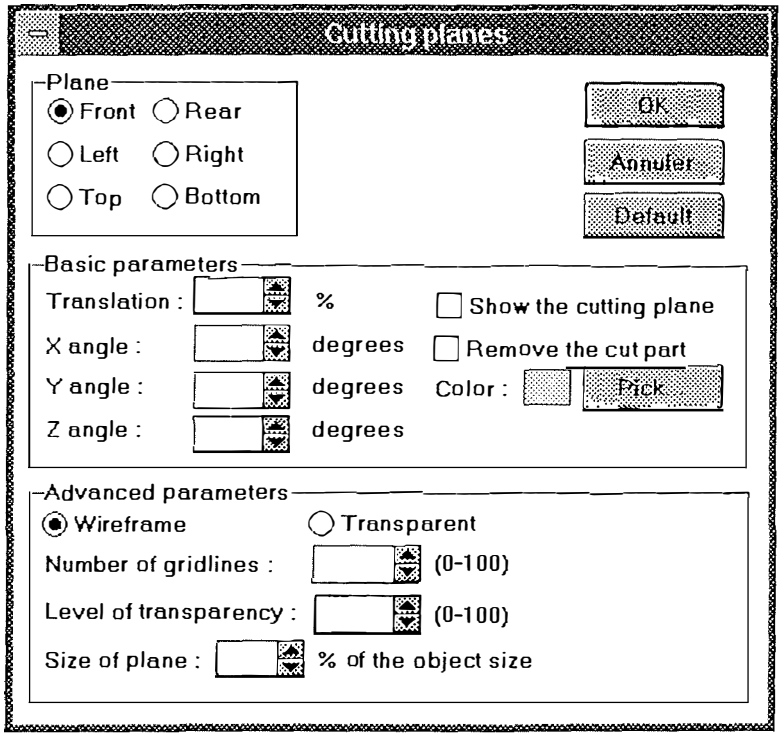


Figure 5-39 : Dialog box proposed by SEGUIA

The group box including the single semantic element "Color" was included in the group box "Basic parameters" when using SEGUIA.

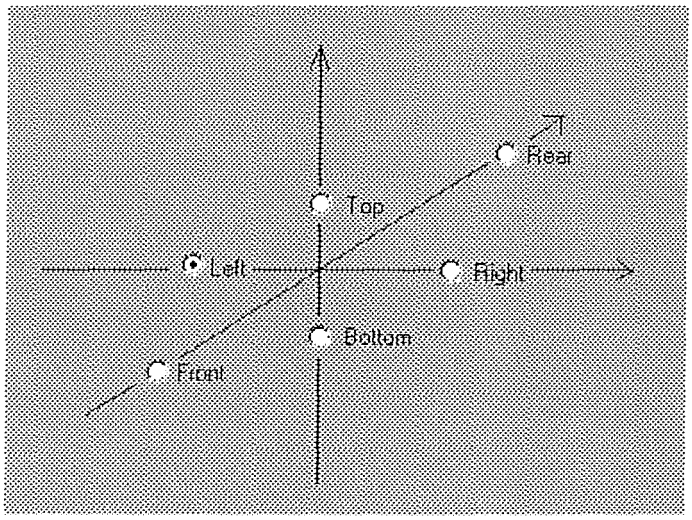


Figure 5-40 : Another way to place CIOs to add meaning

Another big difference is that SEGUIA, as well as TRIDENT, does not take into account linked elements. For example, in Figure 5-38, when the radio button "wireframe" is checked, the linked CIOs "Number of gridlines :" label , the spin button and the "(0-100)" label are activated, and the CIOs relative to Transparency are deactivated. SEGUIA does not include such a semantic and therefore this semantic is not represented in the dialog box. The conversation dimension should include the description of the activation and deactivation of the CIOs. Another way to place these

CIOs is shown in Figure 5-41. This way eases the understanding since parameters corresponding to each radio button are on a single line. This line is actually a sentence that can be easily understood, but the linked elements still remain undefined.

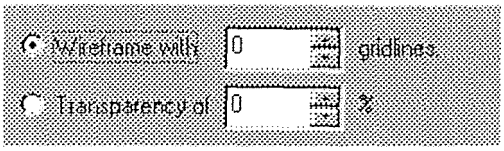


Figure 5-41 : A single line for each option

Another interesting difference is the way buttons are placed. SEGUIA aligned all buttons in relation to the "Pick..." button instead of aligning them along the right side of the dialog box.

This *a posteriori* utilization of the TRIDENT methodological ergonomic rules[31] for CIOs placement proves that it is necessary to keep these rules in mind when designing an interface, especially when speaking about the number of elements in a set (and the problem of choosing the right CIO), but this also brings to the fore some gaps in TRIDENT and SEGUIA like the problem of linked elements. However, the TRIDENT framework and SEGUIA work fine for most of the dialog boxes.

8. Conclusion

8.1 Critic

We remind you that most sub-tasks of the interactive task consist in modifying the current scene and in displaying it when the modification is done. As sub-tasks are linked to tools (Cf. *Chapter 4*), the last sentence can be transformed in : each time a tool has been applied to a scene, the latter must be displayed. It is the reason why in most of the ACGs — the ones that modify the current scene — you find the three elements shown in Figure 5-2.

Concretely, these elements are represented by a physical window whose content is the graphical representation of the scene. The problem we faced was to choose how to logically gather them :

- Within a presentation unit (PU0), independently of the rest of the ACGs (Figure 5-42). In this case, in the dialogue specification (fourth dimension), it must be specified that this PU is called

[31] SEGUIA uses these rules to build the interface

just after most of the other PUs. This PU has only one window (W0) that encompasses the three elements.

- Within a window (W0) which is contained by a presentation unit (PUj), the last one having other windows (Figure 5-43). In the dialogue specification, it must be specified that this window is called just after most of the other windows.

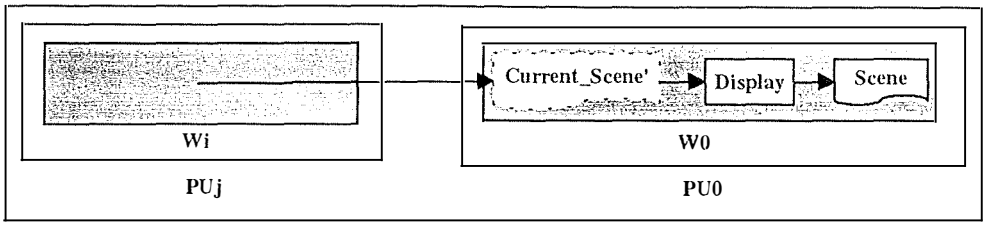


Figure 5-42 : Permanent window : solution 1

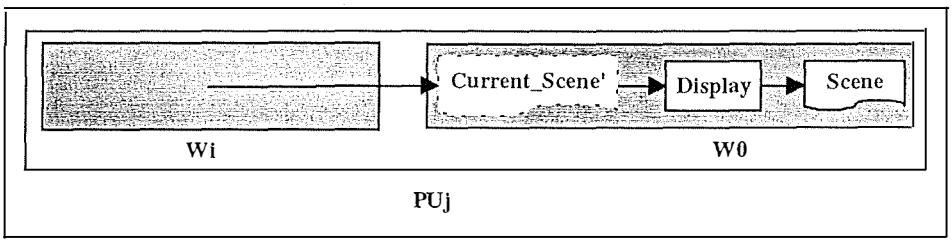


Figure 5-43 : Permanent window : solution 2

We opted for the second solution because it emphasizes the sub-task task progression :

1. Parameters setting and application of the chosen tool (Wi).
2. Displaying of the new scene (W0).

As explained in [BODART95a], each sub-task of the interactive task is mapped into a presentation unit — in our case, that is the same as saying that each tool is mapped into a presentation unit —. So, in most of the presentation units, there is the same logical window W0. The solution we have chosen (Figure 5-43) and this principle can pose us a problem during the control objects hierarchy construction (Cf. *Chapter 6*) among others about the hierarchy of control objects relative to windows. The control object relative to the window W0 (CO-W0) will be the child of several parent control objects. The problem will be analyzed more in depth in Chapter 6.

A way to avoid the last question was to choose the first solution (Figure 5-42) : a presentation unit with only one logical window that contains the three elements quoted above. The control object relative to the window W0 would be the child of only the control object relative to the unique presentation unit (PU0) containing this window. We are convinced that this solution is only

useful to solve that problem but we would lost semantic about the sub-tasks. In our opinion, most of the sub-tasks consist in **modifying** the current scene **and**, then, **displaying** the modified scene.

This window W0 identified in the presentation units is physically implemented as a **permanent window** where only the content is displayed each time the scene has been modified. The TRIDENT methodology has not yet proved itself with regard to applications with permanent windows as it is the case for word processors, spreadsheet applications, drawing programs, conception support software's... We are going to enrich the methodology by taking into account the presence of permanent windows in some applications. To generalize, we define the concept of **central element** as being the content of a permanent window on which tools made available by the application are applied by the users. We illustrate the concept of central element for some software's :

- Word processor → text document.
- Spreadsheet program → spreadsheet.
- Drawing program → drawing sheet.
- ...

The user can manipulate several scenes, each one contained in a different physical permanent windows. That is what we call a MDI (**M**ultiple **D**ocuments **I**nterface) application. So the logical window W0 represents in fact the occurrence of the window displaying the current scene.

8.2 TRIDENT methodology enrichment

We are still considering the case of a weakly structured task and we presume that the task analysis has been done beforehand, with as result the ACGs, each of them corresponding to a sub-task of the interactive task (i.e. to a tool from the toolbox). Moreover, we suppose that the application has at least a central element on which tools are applied. The presentation design will be based on the one proposed in the TRIDENT methodology.

1. Identify the presentation units (PUs). The identification criterion is one PU per sub-task. It is the same as saying one PU per tool or one PU per ACG. A semantic grouping of several ACGs into a PU can be made (Cf. the PU 16 that groups the ACGs 16 and 17, second dimension).

2. Inside each PU, identify the logical windows. We suggest an elimination method of windows identification. First identify the permanent window (in our case W0) that will have the central element. Then group the other messages and functions inside logical windows. It is possible to choose one of the identification criteria proposed in [TAES94] to apply on the rest of the presentation unit.

3. For each logical window, identify the AIOs. First of all, select the window (physical window, dialog box or panel) that will stand for the logical window currently analyzed. Usually, the tools parameters setting will be done via dialog boxes whereas the permanent windows with the central elements will be represented by physical windows. Secondly, if the window does not display the central element, select the right AIOs for each external information in input or output of functions in the logical window. The selection will be applied in function of AIOs selection tables (Cf. Table 5-1, Table 5-2 and Table 5-3), created according to principles of cognitive psychology.

4. Transform of the AIOs into CIOs.

5. Place CIOs.

6. Edit manually the presentation.

Chapter 6
Third dimension :
The software architecture derivation

1. Introduction

Again, we apply the third dimension of the TRIDENT methodological framework *a posteriori* of the program implementation. The object of this part is to see if it is possible to automatically derive an architecture skeleton for our program as it is the case for business oriented software's. The architecture should respect quality criteria : high internal cohesion, weak coupling and independent components. We will follow the steps suggested by [BODART95a] and [BODART95b].

We will (1) theoretically describe the architecture elements and explain how to derive them. Then (2) we will try to establish the architecture for our software.

2. Architecture theoretical description

The architecture consists in a hierarchy that should match the following assumptions :

- "*Each hierarchy element should be derived — directly or indirectly — from task analysis*", [BODART95b].
- "*Autonomy — as perfect as possible —, which is similar to separation, should coexist between elements representing application components and user interface (UI) components. At the level of UI components, this autonomy is prolonged between components realizing the conversation (the dynamic behavior) and components realizing presentation (static appearance)*", [BODART95b].

The autonomy required is possible for business oriented applications where there is no semantic role for the interaction objects. The question we raised was : "*does this autonomy still exists in 3D Viewer software ?*". We first answered that this autonomy was possible with regard to elements concerning the tools (i.e. the dialog boxes and their interaction objects) — if we compare the application to a toolbox — where the interaction objects have no semantic role like it is the case for business-oriented applications. As far as the permanent windows are concerned, we thought that the autonomy was not achievable because the content of these windows (the pieces of bones, their color, their position, the scene characteristics...) had a semantic role for the users. We went the wrong way. The right question that we should have asked is : "*have the contents of the permanent windows an effect on the program behavior ?*". That is to say does the way a scene is composed influence the number of tools made available by the software, the way the program permits the users to set them or the way dialogues are carried out ? In fact there was a misunderstanding about the notion of semantic role.

At first sight we interpreted this notion as a "*semantic role for the user*" whereas it must be interpreted as a "*semantic role for the application*". The semantic role for the user means that the user's behavior depends upon the content of the permanent window which is generally true for any AIO whereas the semantic role for the program means that the application behavior relies on the permanent window content. In our case, whatever the scene looks like or is tuned, the program behavior remains the same. We illustrate the presentation and the functional autonomy by comparing the permanent window with an AIO : the edit box. The edit box is an AIO with as content a string. The permanent window is an AIO with as content a scene. The string of the edit box can be the input or the output of a function that is independent from the edit box presentation. Similarly, the scene of the permanent window is the output of the Display function (see *Chapter 4*), which is independent from the physical window that contains the scene. So, the necessary autonomy is achieved.

The hierarchy model, composed of three generic classes, is presented in Figure 6-1 taken from [BODART95b], where :

- CO stands for **C**ontrol **O**bjects class. It is a generic class which decomposes itself into different types of COs that manage the dialogue and assures that the application data structure is independent from the presentation one.
- AO stands for **A**pplication **O**bjects class. It is a generic class which can not be decomposed and its elements represent the functions of the application.
- IO stands for **I**nteraction **O**bject class. It is a generic class which decomposes itself into two types of OIs : the application dependent CIOs that translate the input/output information for functions and the application independent CIOs that are induced from the dialogue.

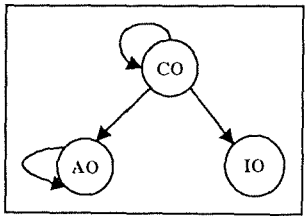


Figure 6-1 : Generic scheme of the architecture model

The rules about the behavior of the three object classes are identical and, also, identical relationships link any pair of these objects in the hierarchy. Each object is an *agent* (Cf. [BODART95b]) and the "uses" relationship that is materialized by an arrow from a parent object to a child object means that :

- The parent object uses the primitives associated to a child object in order to get the needed information for the next interaction step.
- The child object sends events corresponding to significant steps of its behavior to the parent object.

We will now review each objects hierarchy and explain more in depth how they are constructed. When all the hierarchies are complete, the final step consists in integrating all of them in only one hierarchy that represents the global architecture skeleton proposed by the TRIDENT methodology. For legibility reasons, we will not integrate them in this text.

2.1 Application Objects (AO)

Each function in the ACG has a corresponding application object. The AO should be implemented in an object oriented language and there should be an AO for each function of the application. In the hierarchy, each AO is always the child object of only one control object that is responsible for carrying out the function to which it corresponds.

2.2 Control Objects (CO)

Each step in the presentation composition has a corresponding control object and the COs are organized in a hierarchy shown in Figure 6-2. The presentation composition is presented in this way : windows are identified, they are dynamically linked into PUs and these PUs realize the context for execution of the interactive task. The COs hierarchy is constructed in function of the steps brought to the fore just above. The following are the control objects identified in the hierarchy :

- CO-TI : CO corresponding to the interactive task.
- CO-PU : CO corresponding to the presentation units.

- CO-W : CO corresponding to the windows.
- CO-Fc : CO corresponding to the application functions.

Each control object linked to an application function (CO-Fc) is the child object of a control object connected to the window (CO-W) that contains the CIOs used to trigger these function.

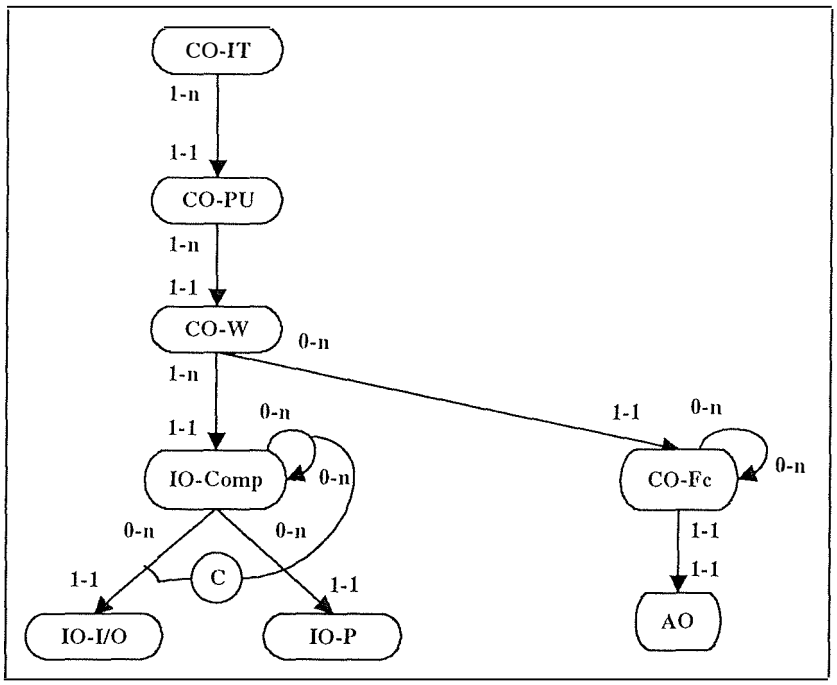


Figure 6-2 : Control objects hierarchy

2.3 Interaction Objects (IO)

The presentation is structured into simple and composite IOs. Composite IOs (IO-Comp, Figure 6-2) are the CIOs corresponding either to windows (like dialog boxes, physical windows or panels) or to simple IOs grouping (child dialog boxes, group boxes). The composite IOs use simples IOs, which constitute the presentation interface. They are input-output interaction objects (IO-I/O, Figure 6-2) like edit box, radio buttons... or they are presentation induced interaction objects (IO-P, Figure 6-2) like push buttons, icons... The IOs can be selected by an expert system in function of several parameters and correspond to CIOs proposed by specific physical environment such as MS-Windows and others. Every IO is a child of a window object control (CO-W).

3. Hierarchies construction

In this dimension, the **continuity** of the TRIDENT methodology is materialized by the use of the task analysis, the ACG and the presentation content. The following objects of the three kinds has to be created :

- A control object corresponding to the interactive task (CO-IT)
- A control object corresponding to each presentation unit (CO-PU_1, ..., CO-PU_n)
- A control object corresponding to each window of each presentation unit (CO-PU_1W_1, ..., CO-PU_1W_m, ..., CO-PU_nW_1, ..., CO-PU_nW_p)
- A control object for each function in the ACG (CO-Fc)
- An interaction object for each input/output information for all functions (IO-I/O)
- An interaction object for each presentation induced object (IO-P)

3.1 Primary hierarchy of functional objects (CO-Fc)

To construct this hierarchy, the ACG obtained from the task analysis is needed. "*Each function in the ACG is mapped onto a functional CO with "uses" relationships between them according to the special property : the primary hierarchy of function COs is quite the inverse hierarchy of the ACG*", as explained in [BODART95b]. Since the ACGs were created by distinguishing tools, the hierarchy will be built by distinguishing the same tools.

3.1.1 Creation of a new scene

Figure 6-3 is showing the primary hierarchy of functional objects for the tool "Creation of a new scene".

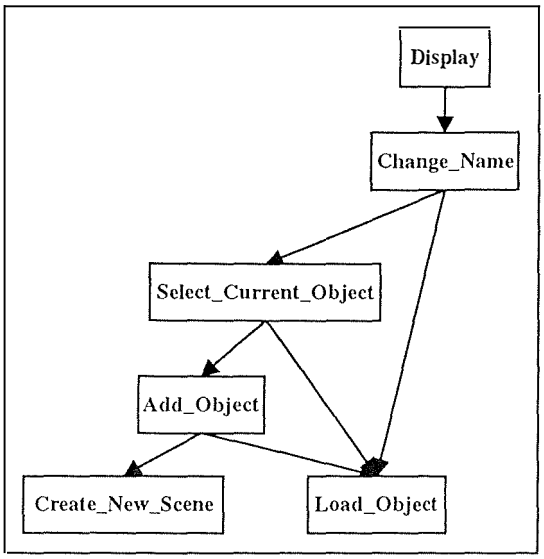


Figure 6-3 : Hierarchy of functional objects for the tool "Creation of a new scene"

3.1.2 Selection of the current scene

Figure 6-4 is showing the primary hierarchy of functional objects for the tool "Selection of the current scene".

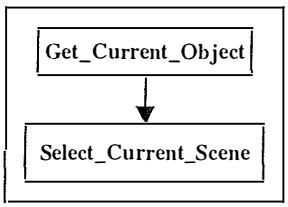


Figure 6-4 : Hierarchy of functional objects for the tool "Selection of the current scene"

3.1.3 Removal of the current scene

Figure 6-5 is showing the primary hierarchy of functional objects for the tool "Removal of the current scene".

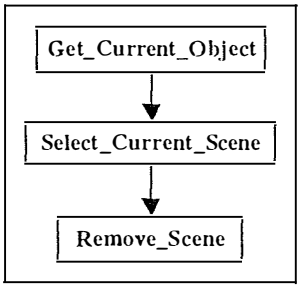


Figure 6-5 : Hierarchy of functional objects for the tool "Removal of the current scene"

3.1.4 Specifying the parameters of the current scene

Figure 6-6 is showing the primary hierarchy of functional objects for the tool "Specifying the parameters of the current scene".

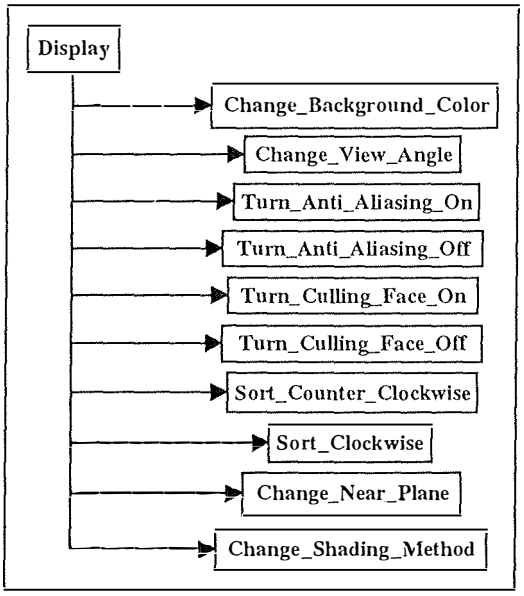


Figure 6-6 : Hierarchy of functional objects for the tool "Specifying the parameters of the current scene"

3.1.5 Geometrical transformation of all the objects in the current scene

Figure 6-7 is showing the primary hierarchy of functional objects for the tool "Geometrical transformation of all the objects in the current scene".

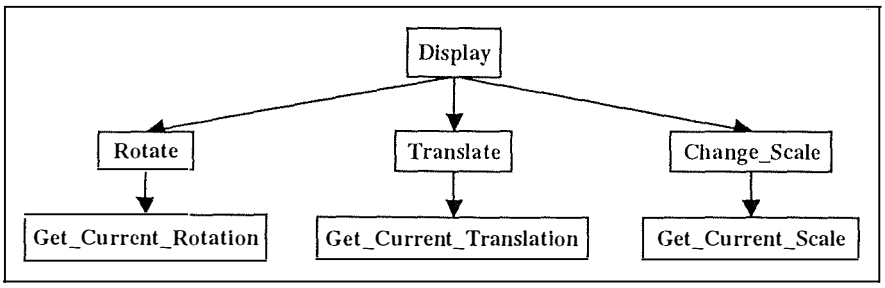


Figure 6-7 : Hierarchy of functional objects for the tool "Geometrical transformation of all the objects in the current scene"

3.1.6 Cutting a part of the current scene

Figure 6-8 is showing the primary hierarchy of functional objects for the tool "Cutting a part of the current scene".

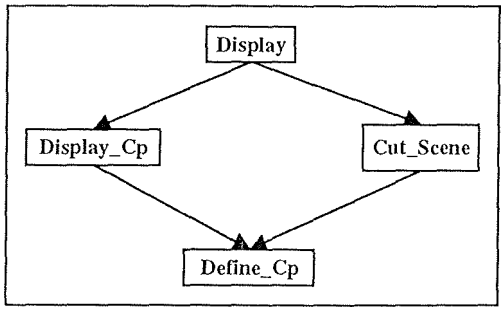


Figure 6-8 : Hierarchy of functional objects for the tool "Cutting a part of the current scene"

3.1.7 Management of the lights

Figure 6-9 is showing the primary hierarchy of functional objects for the tool "Management of the lights".

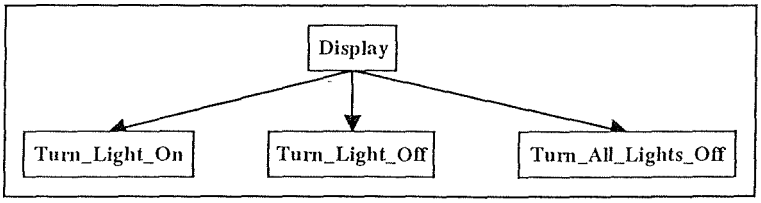


Figure 6-9 : Hierarchy of functional objects for the tool "Management of the lights"

3.1.8 Saving into VRML format

Figure 6-10 is showing the primary hierarchy of functional objects for the tool "Saving into VRML format".

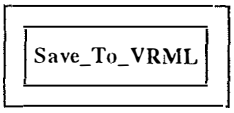


Figure 6-10 : Hierarchy of functional objects for the tool "Saving into VRML format"

3.1.9 Addition of an object into the current scene

Figure 6-11 is showing the primary hierarchy of functional objects for the tool "Addition of an object into the current scene".

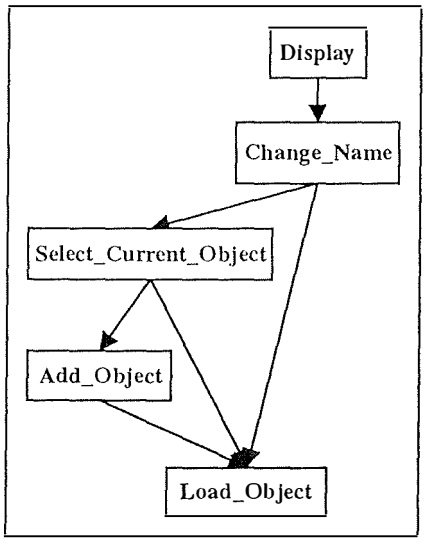


Figure 6-11 : Hierarchy of functional objects for the tool "Addition of an object into the current scene"

3.1.10 Selection of the current object

Figure 6-12 is showing the primary hierarchy of functional objects for the tool "Selection of the current object".

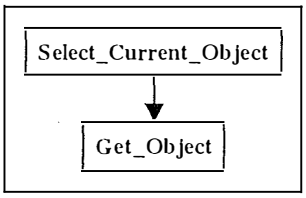


Figure 6-12 : Hierarchy of functional objects for the tool "Selection of the current object"

3.1.11 Removal of the current object from the current scene

Figure 6-13 is showing the primary hierarchy of functional objects for the tool "Removal of the current object from the current scene".

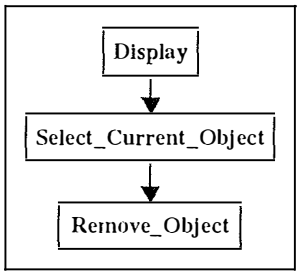


Figure 6-13 : Hierarchy of functional objects for the tool "Removal of the current object from the current scene"

3.1.12 Changing the name of the current object

Figure 6-14 is showing the primary hierarchy of functional objects for the tool "Changing the name of the current object".

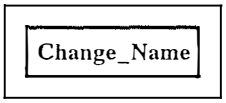


Figure 6-14 : Hierarchy of functional objects for the tool "Changing the name of the current object"

3.1.13 Getting information about an object

Figure 6-15 is showing the primary hierarchy of functional objects for the tool "Getting information about an object".

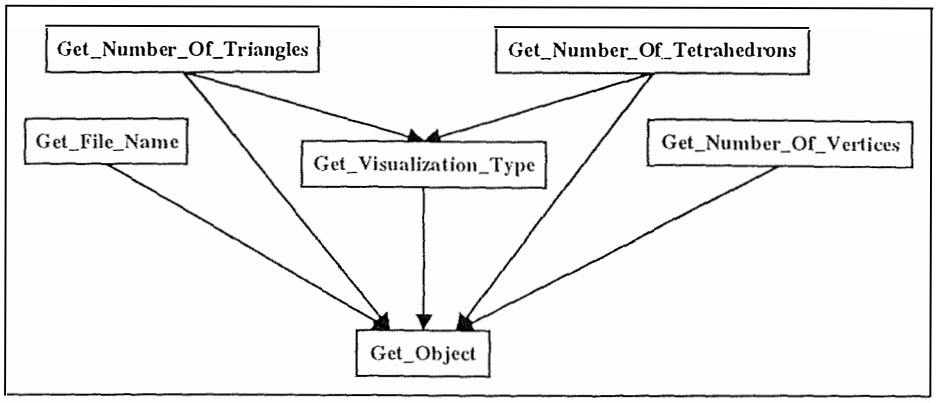


Figure 6-15 : Hierarchy of functional objects for the tool "Getting information about an object"

3.1.14 Changing the color of the current object

Figure 6-16 is showing the primary hierarchy of functional objects for the tool "Changing the color of the current object".

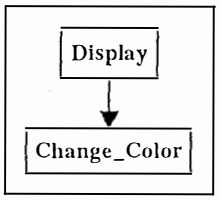


Figure 6-16 : Hierarchy of functional objects for the tool "Changing the color of the current object"

3.1.15 Changing the type of visualization of the current object

Figure 6-17 is showing the primary hierarchy of functional objects for the tool "Changing the type of visualization of the current object".

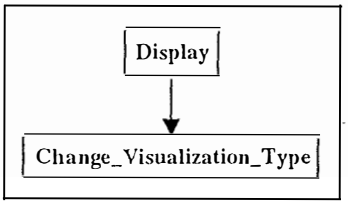


Figure 6-17 : Hierarchy of functional objects for the tool "Changing the type of visualization of the current object"

3.1.16 Showing the current object axis

Figure 6-18 is showing the primary hierarchy of functional objects for the tool "Showing the current object axis".

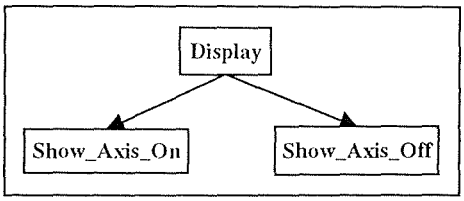


Figure 6-18 : Hierarchy of functional objects for the tool "Showing the current object axis"

3.1.17 Showing the current object box

Figure 6-19 is showing the primary hierarchy of functional objects for the tool "Showing the current object box".

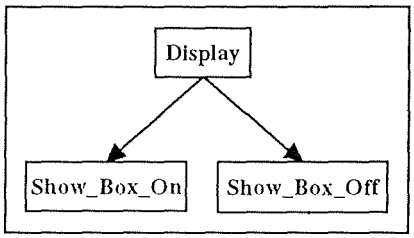


Figure 6-19 : Hierarchy of functional objects for the tool "Showing the current object box"

3.2 Primary hierarchy of control objects relative to the task (CO-IT, CO-PU and CO-W)

Figure 6-21 is showing the primary hierarchy of control objects relative to the task "visualization of 3D objects with the intention of posing a diagnosis". We faced a problem with the permanent window W0 that is present in most of the presentation units and, as a consequence, the control object relative to this window (CO-W0) is a child of several parents. Should this control object be in the hierarchy in as many occurrences as there are occurrences of the permanent window W0 in the presentation units ? We imagined that it was possible to represent the control object CO-W0 in a unique occurrence with "uses" links between the control objects relative to the presentation units the window is belonging to and the unique occurrence of the control object CO-W0 (Cf. Figure 6-20). Nevertheless, in Figure 6-21 this control object is represented several times to make the hierarchy legible.

Another problem has appeared : when to create an occurrence of the control object CO-W0 relative to an occurrence of the permanent window W0 ? The TRIDENT methodology does not tackle the problem of objects creation and deletion. It was presumed that an object was created by his parent object when it was used for the first time and it was destroyed by the same parent object when it was no longer used. The problem with the control object relative to the permanent window is coming from the fact that it is used by several parent objects. Which one of them is responsible for the creation and the deletion of the occurrences of this control object ? In our case, the CO-PU1 parent object is in charge of creating new occurrences of the CO-WO object and CO-PU3 is responsible for the deletion of these occurrences previously created.

We are going to enrich the hierarchy by adding objects creation and deletion constraints when the objects are children of several parents (multi-parents objects):

- Constraint 1 : *CO-PU1* is responsible for the creation of occurrences of *CO-W0.*
- Constraint 2 : *CO-PU3* is responsible for the deletion of occurrences of *CO-W0.*

These constraints only specify which control objects are responsible for the creation and deletion of multi-parents objects but nothing is said about when these operations are carried out.

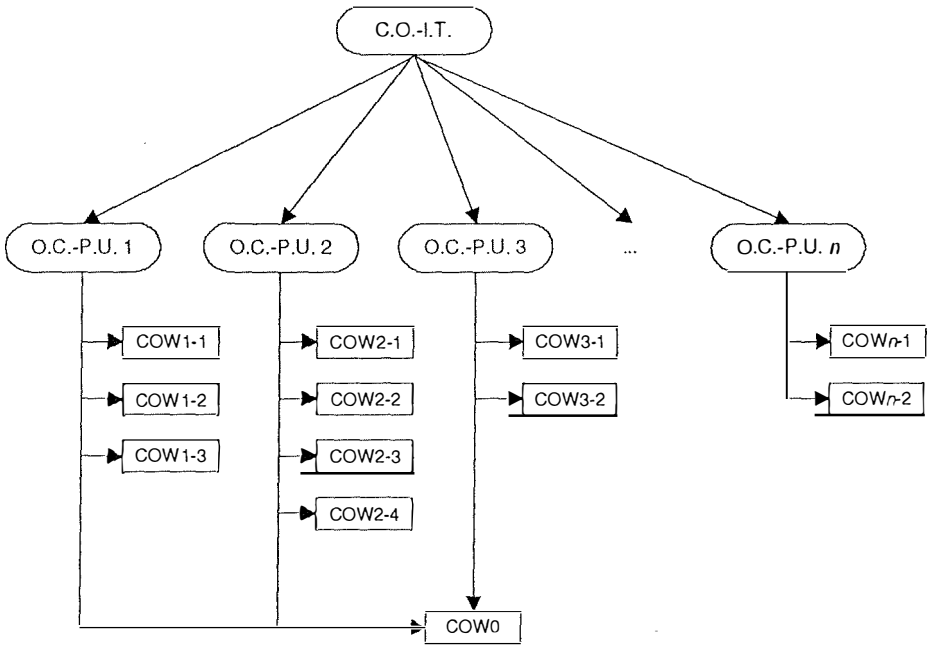


Figure 6-20 : Primary hierarchy of control objects relative to the task : one occurrence of CO-W0

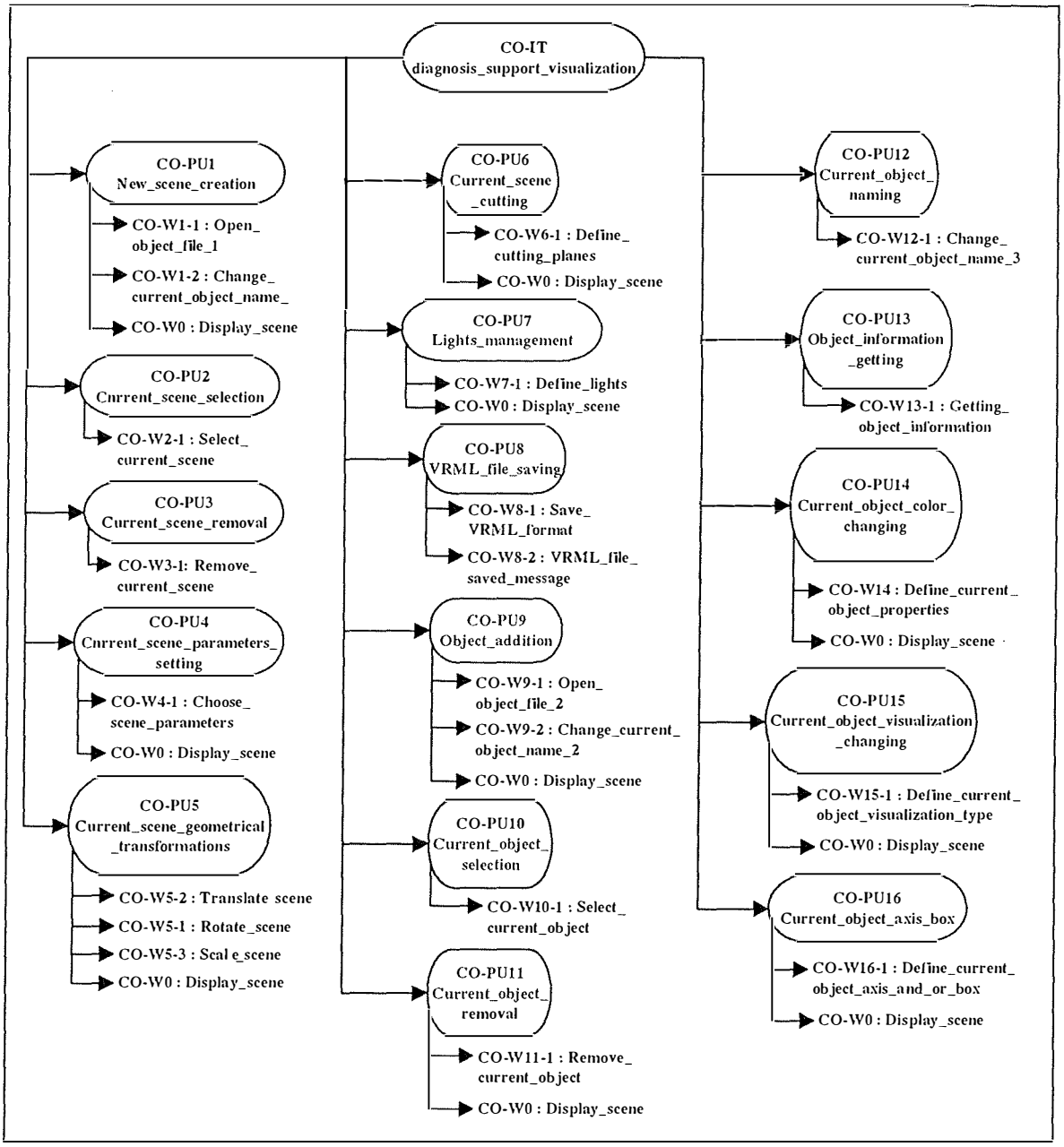


Figure 6-21 : Primary hierarchy of control objects relative to the task : several occurrences of CO-W0

3.3 Primary hierarchy of interaction objects (IO-I/O, IO-P)

In this section is described the primary hierarchy of interaction objects. To facilitate the comprehension, the hierarchy will be decomposed for each window of the application. The name of each AIO begins with one of the prefixes listed below to help to identify which AIO it is.

- CHX → Check box.
- CMN → Cascading menu.
- DBX → Dialog box.
- DLB → Drop-down list box.
- EBX → Single-line edit box.
- FOD → File open standard dialog box.
- FSD → File save standard dialog box.
- GBX → Group box.
- ICO → Icon.
- ITEM → Menu item.
- LBL → Static text.
- LBX → List box.
- MNB → Menu bar.
- PBT → Command button.
- PRO → Progression indicator.
- RBX → Radio button.
- SPB → Spin button.
- TLB → Toolbox.
- WIN → Window.

3.3.1 Main window

WIN_MainWindow
    MNB_Main_Window
        CMN_File
            ITEM_Open
            ITEM_Close
            ITEM_Export
            ITEM_Exit
        CMN_View

ITEM_ Toolbar
ITEM_ Status Line
ITEM_ QuickTools
ITEM_ Objects Toolbar
CMN_Objects
ITEM _ Select_Objects
ITEM _ Change_Name
ITEM _ Object_Type
ITEM _ Axis_and_Box
ITEM _ Object_Properties
CMN_Scene
ITEM _ Add_Object
ITEM _ Remove_Object
ITEM _ Background_Color
ITEM _ Set_Angle
ITEM _ Set_Scale
ITEM _ Translate
ITEM _ Cutting_Planes
ITEM _ Lights
ITEM _ Options
CMN_ Window
ITEM _ New_Window
ITEM _ Cascade
ITEM _ Tile
ITEM _ Arrange_Icons
ITEM_window_name_1
...
ITEM_window_name_n

CMN_Help
ITEM _ About_3D_Viewer
ITEM _ Data_Information
TLB_Shortcut_Tools
PBT_Show_ Current_Object_Axis
PBT_Rotate_Scene_Along_Y_Axis
PBT_Rotate_Scene_Along_X_Axis
PBT_Rotate_Scene_Along_Z_Axis
PBT_Initial_Rotation_State
PBT_Change_Current_Object_Color

PBT_Turn_Light_On_Off
PBT_Translate_Scene_Right
PBT_Translate_Scene_Left
PBT_Zoom_Scene_In
PBT_Zoom_ Scene_Out
TLB_Non_Shortcut_Tools :
PBT_Add_Object
PBT_Continuous_Rotation_Speed_Increase
PBT_Continuous_Rotation_Speed_Decrease
PBT_Change_Current_Object_Color
PBT_Manage_The_Lights
PBT_Manage_Current_Object_Axis
PBT_Manage_Scene_Translation
PBT_Manage_Scene_Scaling
PBT_Manage_Scene_Rotation
PBT_Create_New_Current_Scene_View
TLB_Current_Object
DLB_Current_Object
PBT_Remove_Current_Object
PBT_Rename_Current_Object
PBT_Add_New_Object

We bring to the fore that the *ITEM_window_name_1*, ..., *ITEM_window_name_n* menu items from the *CMN_ Window* menu are in fact the names of the different permanent windows that exist at a certain point. They allow users the select the current permanent window on which they want to work.

3.3.2 Window "Display_scene" (W0)

The only element of the interaction objects hierarchy for the window containing the scene is :

WIN_W0

3.3.3 Window "Open_object_file_1" (W1-1)

FOD_W1-1
EBX_File_Name
PBT_Ok
PBT_Cancel

...

The interaction objects hierarchy for the window W1-1 is not complete. Standard open file windows usually have more AIOs but are listed in the hierarchy only the most necessary one.

3.3.4 Window "Change_current_object_name_1" (W1-2)

DBX_W1-2
- EBX_New_Object_Name
- PBT_Ok
- PBT_Cancel

3.3.5 Window "Select_Current_Scene" (W2-1)

There is no interaction objects hierarchy for this logical window since it is only represented by an ITEM_Window_Name_i menu item from the CMN_Window_ menu.

3.3.6 Window "Remove_current_scene" (W3-1)

There is no interaction objects hierarchy for this logical window since there is no AIO representing this logical window.

3.3.7 Window "Choose_scene_parameters" (W4-1)

DBX_W4-1
- GBX_Size
  - SPB_Near_Plane
  - SPB_View_Angle
- GBX_Shading_Method
  - RBX_Flat_Shading
  - RBX_Smooth_Shading
- GBX_Vertices_Sorting_Method
  - RBX_Clockwise_Sorting
  - RBX_Counter_Clockwise_Sorting
- GBX_Background_Color
  - PBT_Pick_Background_Color
  - ICO_Background_Color
- GBX_Other

CHX_Antialiasing
CHX_Culling_Face
PBT_Ok
PBT_Cancel

3.3.8 Window "Rotate_scene" (W5-1)

DBX_W5-1
GBX_New_Angles
SPB_New_X_Angle
SPB_New_Y_Angle
SPB_New_Z_Angle
GBX_Current_Angles
EBX_Current_X_Angle
EBX_Current_Y_Angle
EBX_Current_Z_Angle
PBT_Ok
PBT_Cancel
PBT_Initial_View

3.3.9 Window "Translate_scene" (W5-2)

DBX_W5-2
GBX_New_Position
SPB_New_X_Position
SPB_New_Y_Position
SPB_New_Z_Position
GBX_Current_Position
EBX_Current_X_Position
EBX_Current_Y_Position
EBX_Current_Z_Position
GBX_Translation_Method
RBX_Best_Fit
RBX_Absolute
RBX_Relative
PBT_Ok
PBT_Cancel
PBT_Center

3.3.10 Window "Scale_scene" (W5-3)

DBX_W5-3
- GBX_New_Scale
  - SPB_New_X_Scale
  - SPB_New_Y_Scale
  - SPB_New_Z_Scale
  - CHX_Keep_Aspect_Ratio
- GBX_Current_Scale
  - EBX_Current_X_Scale
  - EBX_Current_Y_Scale
  - EBX_Current_Z_Scale
- PBT_Ok
- PBT_Cancel

3.3.11 Window "Define_cutting_planes" (W6-1a and W6-1b)

DBX_W6-1a
- GBX_Planes_Parameters
  - DLB_Cutting_Plane
  - CHX_Show_Cutting_Plane
  - CHX_Cut_Scene
  - SPB_Distance
  - SPB_X_Angle
  - SPB_Y_Angle
  - SPB_Z_Angle
- GBX_Other
  - PBT_Pick_Plane_Color
  - ICO_Plane_Color
- PBT_Ok
- PBT_Cancel
- PBT_Default
- PBT_Advanced_>>

DBX_W6-1b
- GBX_Planes_Parameters
  - DLB_Cutting_Plane
  - CHX_Show_Cutting_Plane

CHX_Cut_Scene
SPB_Distance
SPB_X_Angle
SPB_Y_Angle
SPB_Z_Angle
GBX_Other
PBT_Pick_Plane_Color
ICO_Plane_Color
GBX_Advanced
RBX_Wireframe_Plane
SPB_Number_Of_Wires
RBX_Translucent_Plane
SPB_Translucence_Percentage
SPB_Plane_Size
PBT_Ok
PBT_Cancel
PBT_Default
PBT_<<_Advanced

3.3.12 Window "Define_lights" (W7-1)

DBX_W7-1
GBX_Lights
DLB_Lights
CHX_On/off
GBX_Components
PBT_Pick_Light_Color
ICO_Light_Color
SPB_Specular_Component
ICO_Specular_Component
GBX_Position
EBX_Light_X_Position/direction
EBX_Light_Y_Position/direction
EBX_Light_Z_Position/direction
CHX_Near
CHX_All_Lights_Off
PBT_Ok
PBT_Cancel
PBT_Default

3.3.13 Window "Save_VRML_format" (W8-1)

FSD_W8-1
- EBX_File_Name
- PBT_Ok
- PBT_Cancel
- ...

The interaction objects hierarchy for the window W8-1 is not complete. Standard save file windows usually have more AIOs but are listed in the hierarchy only the most necessary one.

3.3.14 Window "VRML_file_saved_message" (W8-2)

DBX_W8-2
- PRO_File_Saving
- PBT_Cancel

3.3.15 Window "Open_object_file_2" (W9-1)

DBX_W9-1
- EBX_File_Name
- PBT_Ok
- PBT_Cancel
- ...

The interaction objects hierarchy for the window W9-1 is not complete. Standard save file windows usually have more AIOs but are listed in the hierarchy only the most necessary one.

3.3.16 Window "Change_current_object_name_2" (W9-2)

DBX_W9-2
- EBX_Object_Name
- PBT_Ok
- PBT_Cancel

3.3.17 Window "Select_current_object" (W10-1)

DBX_W10-1

GBX_Objects
LBX_Objects
PBT_Ok
PBT_Cancel

3.3.18 Window "Remove_current_object" (W11-1)

DBX_W11-1
LBL_Confirmation
PBT_Yes
PBT_No

3.3.19 Window "Change_current_object_name_3" (W12-1)

DBX_W12-1
EBX_Object_Name
PBT_Ok
PBT_Cancel

3.3.20 Window "Getting_object_information" (W13-1)

DBX_W13-1
GBX_Objects
LBX_Objects
GBX_Mesh
LBX_Objects
EBX_Number_Of_Elements
EBX_Number_Of_Vertices
EBX_Display_Type
EBX_Object_Name
PBT_Close

3.3.21 Window "Define_current_object_properties" (W14-1)

DBX_W14-1
GBX_Transparency
CHX_Object_Transparent
SPB_Translucence_Percentage

GBX_Colors
PBT_Pick_Object_Color
ICO_Object_Color
SPB_Specular_Component
SPB_Shininess_Percentage
PBT_Ok
PBT_Cancel
PBT_Default

3.3.22 Window "Define_current_object_visualization_type" (W15-1)

DBX_W15-1
GBX_Surface_Display
RBX_Filled_Surface
RBX_Mesh
RBX_Points_Cloud
PBT_Ok
PBT_Cancel
PBT_Default

3.3.23 Window "Define_current_object_axis_and_or_box" (W16-1)

DBX_W16-1
GBX_Axis
CHX_Show_Axis_On/Off
CHX_Move_Axis_On/Off
PBT_Pick_Axis_Color
ICO_Axis_Color
SPB_Axis_Length
GBX_Box
CHX_Show_Box_On/Off
PBT_Pick_Box_Color
ICO_Box_Color
SPB_Box_Size

4. Conclusion

4.1 Critic

From the *a posteriori* application of the third dimension on our application, we noticed that the permanent window identified in previous dimension as W0 and present in several presentation units gave rise to two important questions :

- Is there an independence between the permanent window (i.e. the user interface) and the elements representing the application components (i.e. the functions) ?
- In the control objects hierarchy, is it possible to have a control object (in this case CO-W0, the control object relative to W0) being the child of several parent control objects (the CO-PUs, the control objects relative to the presentation units). If so, which parent objects are responsible for the creation and the deletion of occurrences of the child object ?

The first question already discussed in Section *Architecture theoretical description* is important because the way the TRIDENT methodology establishes a software architecture is based on the assumption that there is an autonomy between elements representing application components and user interface components. We have explained that this autonomy was achieved because the content of the permanent window — that is to say the scene embodying pieces of trabecular bones or other objects — does not modify the program behavior and we have made a comparison between the permanent window represented by the physical window AIO and the edit box AIO to point out that this permanent window AIO is independent from the "Display" function which has as output parameters the content of the permanent window (see *Architecture theoretical description* section above).

The second question arose from the fact that in almost all the PUs — they are corresponding to tools — identified in the second dimension, we have identified a common window W0. Consequently, because of the construction approach of the hierarchy of control objects suggested in Figure 6-2, the control object relative to W0 is the child of several control objects, each one corresponding to a presentation unit encompassing W0. It seems that there is no problem for the CO-W0 to be linked with "uses" relations to several CO-PUs. We have decided, in this text, to represent as many occurrences of the object CO-WO as it is linked to a parent object (Figure 6-21) just for a legible representation of the hierarchy. Such a control object should, in principle, be represented in only one occurrence connected with as many " uses " links as it has parents.
Since it is accepted that an object can be multi-parents (i.e. having several parents), the question of its creation and deletion comes in mind. In the TRIDENT methodology, we always presume that an object is created by its parent when it is used for the first time and it is deleted by the same parent when it is no longer used. In the case of multi-parents object the problem is coming from

the fact that we do not know which object is responsible for the creation or the deletion of them. It is the reason why we added creation and deletion constraints to the hierarchy. See *Primary hierarchy of control objects relative to the task (CO-IT, CO-PU and CO-W)* section for an example of such constraints. Any object can be responsible for the creation and/or deletion of a multi-parents objects even if it is not the parent of the object it is creating and/or deleting. In the same idea, the object that is responsible for the deletion of a multi-parent object is not necessarily the one that is responsible for the creation of the same object.

The architecture used at the present time (cf. Figure 2-48) is different from the one suggested by the application of the TRIDENT methodology. The advantages of establishing an architecture as proposed by the third dimension of the TRIDENT methodology are the one quoted in [BODART95a] and [BODART95b] :

- High internal cohesion
- Weak coupling
- Independent components

Furthermore, all the designers of interactive applications using the TRIDENT methodological framework will use the same architecture skeleton. It is a way to standardize the architectures and this facilitates them to quickly understand the structure of applications made by other designers using the same methodology.

We did not develop the fourth dimension of the TRIDENT methodological framework for two main reasons. First, we looked into it and we did not find anything that could be a problem in relation to applications helping to perform weakly structured tasks and having permanent windows. Secondly, we think that it would have drastically increased the number of pages (because of the great number of presentation units and dialog boxes) without bringing interesting things. However, we do not mean that the fourth dimension is no use.

Nevertheless, we show in Figure 6-22 how the inter-PU dialog would look like for an application with n PUs. Each UP (which corresponds to a tool) is connected with all the other PUs. When we look into the toolbox model shown in Figure 4-1 and we compare it with the structure of the Petri network shown in Figure 6-22, we notice a great similarity. Indeed, the user takes a tool in the menu, then he fills up the parameters in the dialog box corresponding to the selected tool, and finally the tool is applied to the central element of the permanent window and the user has a new opportunity to select any other tool. The behavior would be exactly the same with a toolbox, where the worker would select the right tool and would apply it to the object he manipulates. Then he would put his tool back into his toolbox and would maybe use another tool.

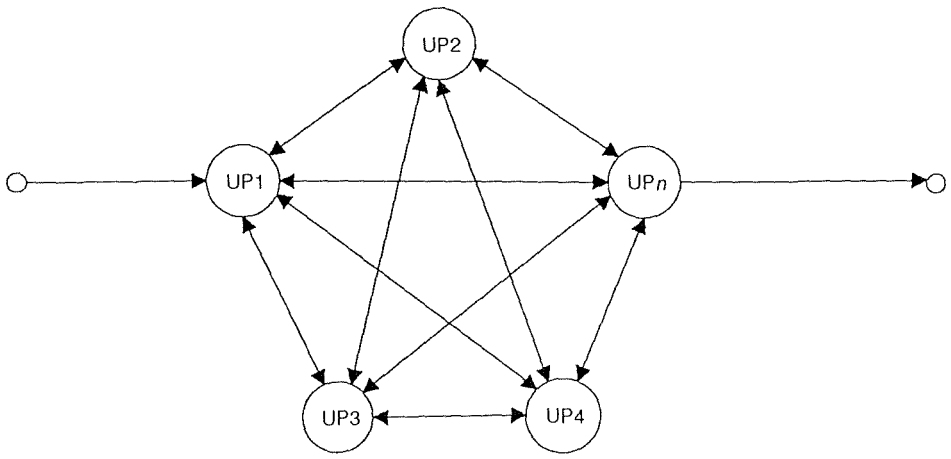


Figure 6-22 : Pietri network for inter-PU dialog specification

4.2 TRIDENT methodology enrichment

The third dimension of the TRIDENT methodology can be applied to automatically derive an architecture skeleton for software having the following characteristics :

- They help to carry out a weakly structured type of task (prescribed and/or decision making)
- They have at least one permanent window
- They are based on the toolbox model

If the following assumption is verified :

- The content of the permanent window(s) has no influence on the program behavior, has no semantic role.

When the last assumption is not verified, the hypothesis of autonomy between the presentation components and the application components is not respected and so, we cannot tell if it is still possible to automatically generate an architecture.

We suggest the same approach as in the TRIDENT methodology with few modifications :

1. Construct the primary hierarchies of functional objects (CO-Fc). The rule of construction is : each function in the ACGs is mapped onto a functional CO. These COs are linked with a "uses" relationships according to this property : the CO-Fc hierarchy is quite the inverse hierarchy of the ACG. In the first dimension enrichment, we have suggested to establish an ACG for each identified tool. As a consequence, there are as many hierarchies of functional objects as there are ACGs.

2. Construct the primary hierarchy of control objects relative to the task by following the steps suggested by Figure 6-2:

- A CO-IT for the interactive task
- A CO-PU for each presentation unit
- A CO-W for each window

It is possible to represent a multi-parents object. In this case it has to be linked with "uses" relationship to each parent it has and creation/deletion constraints have to be added. These constraints specify which control objects are responsible for the creation and/or the deletion of the multi-parent objects.

3. Construct the primary hierarchies of interaction objects. The rule is : one hierarchy by window.

4. Aggregate all the hierarchies into a unique one. For more information see [BODARD95b].

Chapter 7 : Conclusion

From a critical analysis of the TRIDENT methodological framework applied to a 3D visualization program as a support for diagnosis process, we finally enlarged this critic at a higher level. Indeed, we enlarged the scope for applications corresponding to weakly structured tasks and owning at least one permanent window. We have tried to adapt the TRIDENT methodology for the conception of such applications.

This conclusion is divided in three main parts. First, the *toolbox* model is confirmed for such a kind of applications, then the probable link between weakly structured tasks and applications based on a permanent window is discussed and we finally summarize the TRIDENT methodological framework with enhancements for the design of applications supporting weakly structured tasks and having permanent windows.

The toolbox model, as explained above, seems to suit very well for weakly structured tasks and especially for 3D graphics applications. Indeed, these applications use a central element where each functionality is considered as a tool acting or working on the central element, as shown in Figure 7-1.

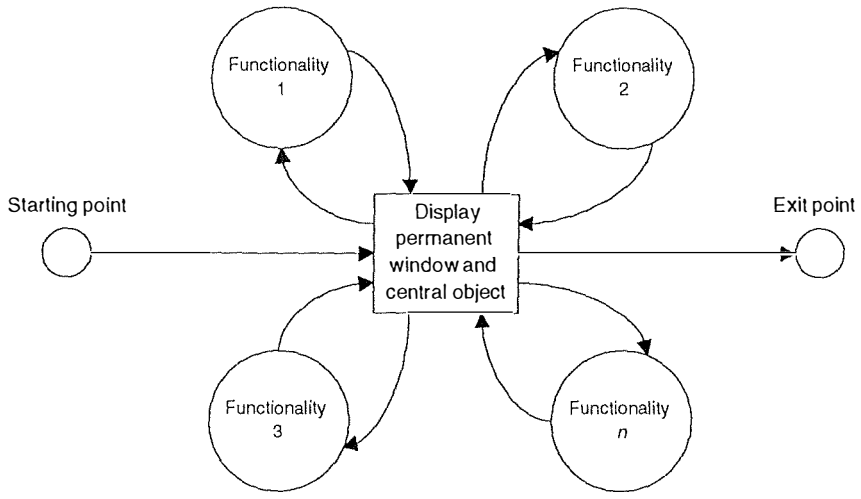


Figure 7-1 : The toolbox model

This model is especially useful when speaking in terms of modifications or enhancements in the application code. Adding a new tool, removing an obsolete or unused tool or modifying an existing tool is almost straightforward as long as it is independant from the other tools.

The conception and implementation approach of our program is essentially based on this characteristic (toolbox model). Since we did not know the task, neither the requirements, which would be difficult to develop a "standard" management program, we started developing a prototype including the central element, in our case, the scene and the objects. Then we added functionality as users were needing them. Actually, the order in which functionality were added is not important, since each one is independent from the other ones. The process of developing such an application is summarized in Figure 7-2.

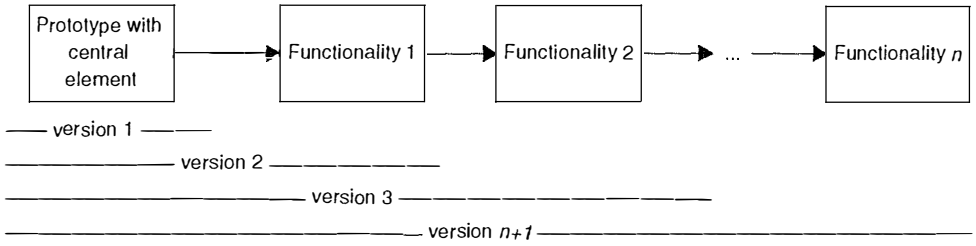


Figure 7-2 : Development process for a toolbox model based application

As it has been discussed in previous chapters, our application, like many different programs as word processors or drawing programs, has a very different characteristic from the common management programs which is its permanent windows. The question we asked is to know if there is or not a link between these permanent, central windows and the weak structure of the task. We said that we have not been able to analyze the task, but we do know that visualization is not a structured task. At any time during this process, we need to see the result of the previous action (feedback), and we then decide the next action to perform. This systematic, necessary feedback justifies a permanent window where the temporary result of modifications and actions applied to the central element can be consulted to allow the user to evaluate progress state of the task, once again, because this one is not predefined and not known. Let us imagine a second that this process would be structured. This would result in a different presentation of the program. Where we have a scheme as shown in Figure 7-1 for unstructured tasks, we would have what is shown in Figure 7-3 for a structured task.

The model shown in Figure 7-3 corresponds to an application where the user enters a whole set of parameters, such as rotation angle, color, ... and where the image is finally displayed. We could imagine that such a process could be used by a very experienced engineer, however, we think that it would prevent him from seeing the impact of *each* parameter which is very important when tuning the right image. Moreover, since each image is different from another one, settings for an image could not suit for another one. So we think that the permanent window has a strong relation with the structure of the task of visualization as a support for diagnostic.

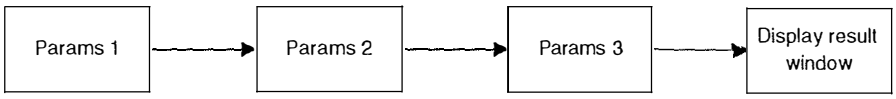


Figure 7-3 : Common dialog box based application

Of course, there are a lot of other software that support unstructured tasks, like word processors for typing a letter, or CAD programs for designing new houses. For these programs, we also find a permanent window, the white sheet or the house. A way to structure tasks performed by these programs would be to restrict the domain, and that what is done with word processors. For example, the Curriculum Vitae Assistant provided with Microsoft Word® is a restriction of the domain "Typing a document". It provides a set of dialog boxes asking a series of questions, ranging from the name and forename of the person to the accomplished studies and hobbies. Then the assistant makes up the data into pages and presents a ready-to-print document, following the model shown in Figure 7-3. We could imagine the same for a company which sells 50 different styles of houses, where the buyer can select the number of rooms, the kind of main door, color of windows, ... and the CAD program would just ask the set of parameters and then builds the house according to the requirements.

However, these programs have to take into account that these *conception* or *decision making* tasks are not well defined and that people can change their mind. In such cases, these program should anyway include a set of tools to allow the user to change few details. So we really think that concepts of permanent window and weakly structured tasks are very close. However, more studies are to be conducted to confirm or not the link.

We did not follow step by step the TRIDENT methodological framework when developing the programs because of the time it would have required. However, we applied it *a posteriori*, and in many respects, the application fits to the framework, but we remarked several differences which are relative to the weakly structured type of the task. We pointed out these differences and tried to solve or propose ideas of solutions. Hereby, we summarized the framework that we suggest to follow when developing applications where the **task is weakly structured**, that includes a **permanent window** containing a **central element.**

1st dimension : Task analysis

Identify the task, goals and sub-goals

Deduct tools from sub-goals and the central elements on which the tools are applied.

Identify procedures

Identify objects and semantic functions from procedures

Build Actions Chaining Graphs

2nd dimension : Presentation design

- Identify Presentation Units (1 Presentation Unit ≡ 1 tool)
- Identify Windows (Identification criterion : elimination)
- AIO selection :
  - 1 tool ≡ 1 dialog box
  - Central element ≡ Physical window if the central element has to be displayed
- Transformation from AIO to CIO
- CIO placement
- Manual edition of the presentation

3rd dimension : Software architecture derivation

- Construct the primary hierarchies of functional objects (one per ACG)
- Construct the primary hierarchy of control objects relative to the task (multi-parents objects are allowed)
- Construct the primary hierarchies of interaction objects
- Aggregate all the hierarchies into a unique one

4rd dimension : Dialogue specification

- Specification of the inter-UP dialogue
- Specification of the inter-windows dialogue
- Specification of the intra-windows dialogue

We insist that this approach we propose is only based on the analysis of one single case : 3D Viewer. It should be validated on more cases to prove feasible or not.

Reference Books

[BODART93] Bodart, F. & Pigneur, Y., "*Conception assistée des systèmes d'information*", MIPS, 1993.

[BODART95a] Bodart, F. & Hennebert, A.-M. & Leheureux, J.-M. & Provot, I. & Vanderdonckt, J. & Zucchinetti, G., "*Dimensions clé pour une méthodologie de développement d'applications interactives*", Namur, 1995.

[BODART95b] Bodart, F. & Hennebert, A.-M. & Leheureux, J.-M. & Provot, I. & Sacre, B. & Vanderdonckt, J., "*Towards a Systematic Building of Software Architecture : the TRIDENT Methodological Guide*", Namur, 1995.

[BRIGGS95] Briggs, T. L., *Computed Body Tomography and Magnetic Resonance Imaging*, MCSP, January 1995.

[BYTE0896] BYTE, *3-D for everyone*, McGraw-Hill Companies, Inc., October 1996.

[DUBOIS96] Dubois, E., Class notes, FUNDP, 1996.

[FAISON94] Faison, T., *Borland C++ 4 Object Oriented Programming, Third Edition*, SAMS Publishing, 1994.

[FROST93] Frost, H.M., *Suggested fundamental concepts in skeletal physiology*, Calc Tiss Int, Vol. 52, pp. 1-4, 1993.

[GOOSSENS95] Goossens, P. and Wauthier, B., *The trabecular bone and morphological Analysis System.: a research and an Educational Training System for Students*, Mémoire, FUNDP, 1995.

[JASC96] JASC Inc., http://www.jasc.com, *Paint Shop Pro v. 4.10*, 1996.

[KELLER95] Keller, T.S. and Hanson, T., *Osteoporosis of the spine*, University of Göteborg, Sweden and University of Vermont, VT, USA, Revision 1, 1995.

[KELLER89] Keller, T.S., Hanson, T., Abram, A.C., Spengler, D.M., Panjabi, M.M., *Regional variations in the compressive properties of lumbar vertebral trabeculae: Effect of disc*

degeneration, Spine, 1989, Vol. 14, pp. 1012-1019.

[KELLER92] Keller, T.S., Moeljanto, E., Main, J.A., Spengler, D.M., *Distribution and orientation of bone in the human lumbar vertebral centrum*, J Spinal Disorders, 1992, Vol. 5, pp 60-74.

[KELLER93] Keller, T.S., ZIV, I., Moeljanto, E., Spengler, D.M., *Interdependance of lumbar disc and subdiscal bone properties: A report of the normal and degenerated spine*, J Spinal Disorders, 1993, Vol. 6, pp. 103-113.

[LECHARLIER95] Lecharlier, B., Class notes, FUNDP, 1995.

[LORENSEN87] Lorensen, W. E. and Cline, H. E., *Marching Cubes: A High resolution 3D Surface Construction Algorithm*, Computer Graphics, 1987.

[MCCORMICK87] McCormick, B.H., DeFanti, T.A. and Brown, M.D., *Visualization in Scientific Computing*", Report of the NSF Advisory Panel on Graphics, Image Processing and Workstations, 1987.

[MEYER88] Meyer, B., *Object-Oriented Software Construction*, Prentice Hall International, 1988.

[MSDEV96] Programming in Visual C++, MFC 4.0, Class Library Référence, Microsoft 1996.

[PICKOVER90] Pickover, C. A., *Computers, patterns, chaos and beauty*, St. Martin's Press, 1990.

[ROSENBLOEM94] Rosenbloem, L. et al., *Scientific Visualization Advances and Challenges*, Harcourt Brace & Company, London, 1994.

[RUMBAUGH91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. And Lorensen, W., *Object-Oriented Modeling and Designing*, Prentice Hall, 1991.

[SCHROEDER96] Schroeder, W., Martin, K. And Lorensen, B., *The Visualization Toolkit, an object oriented approach to 3D Graphics*, Prentice Hall, 1996.

[SEGUIA97] Système Expert pour la Génération d'une "User Interface" Automatique, Vanderdonckt, J., Conception assistée de la présentation d'une IHM ergonomique pour une application de gestion hautement interactive, Thèse de doctorat, FUNDP, July, 9th, 1997.

[SULLIVAN95] Sullivan, J. M. and Zhang, J.Q., *Adaptive Mesh Generation Using a Normal Offsetting Technique*, University of Vermont, VT, USA and Baystate Technologies, MA, USA, 1995.

[TAES94] Taes, F. & Equipe TRIDENT, "*Tâche interactive de l'enregistrement d'un bon de commande*", Etude de cas, Namur, December 1994.

[VANDERDONCKT93a] Vanderdonckt, J., "*Sujet : Révision du document concernant la dérivation de style(s) d'interaction*", rapport de la réunion TRIDENT du 3/11/93, Namur, novembre 1993.

[VANDERDONCKT93b] Vanderdonckt, J., "*A Corpus of Selection Rules for Choosing Interaction Objects*", TRIDENT project, Technical report, Namur, August 1993.

[VRMLSGI] *VRML : Basics*, SGI Inc., http://vrml.sgi.com/basics/, 1997.

[WATT93] Watt, A., *3D Computer Graphics*, Second edition, Addison Wesley, 1993.

[WOLFF1892] Wolff, J., *Das Destez der Transformation der Knochen*, Hirschwald, Berlin, 1892.

[WRIGHT96] Wright, R.S. Jr. and Sweet, M., *OpenGL Superbible*, Waite Group Press, 1996.

[ZETTERBERG90] Zatterberg, C., Mannius, S., Mellström, D. Et al., *Osteoporosis and back pain in the elderly. A controlled epidemologic and radographic study*, Spine, Vol. 15, pp. 783-786, 1990.

[ZETTERBERG94] Zatterberg, C., Sjöstedt, Ä, Ziden, L. et al., *Epidemiology of kip fractures in Göteborg, Sweden, 1940-1991*, Scandinavian Orthopaedic Association, Proceeding of the 47th Assembly, Reykjavik, Iceland, June 8-11. Acta Orthop Scand 1994; 65(suppl260)30.

Dictionaries

[AHD86] The American Heritage Dictionary and Electronic Thesaurus, 1986.

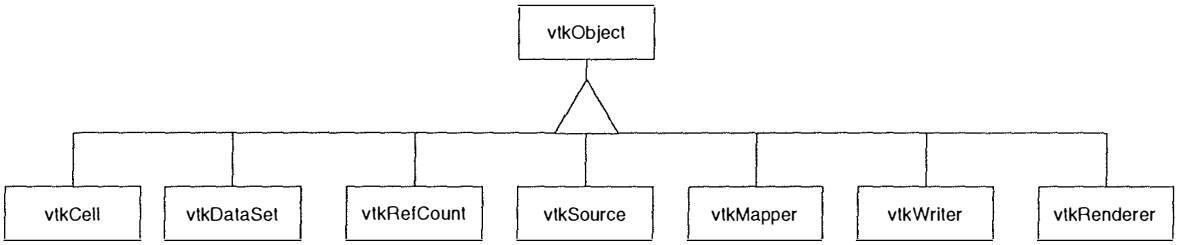
The Collins Electronic Dictionary, 1992.

HARRAP'S Dictionary, 1993.

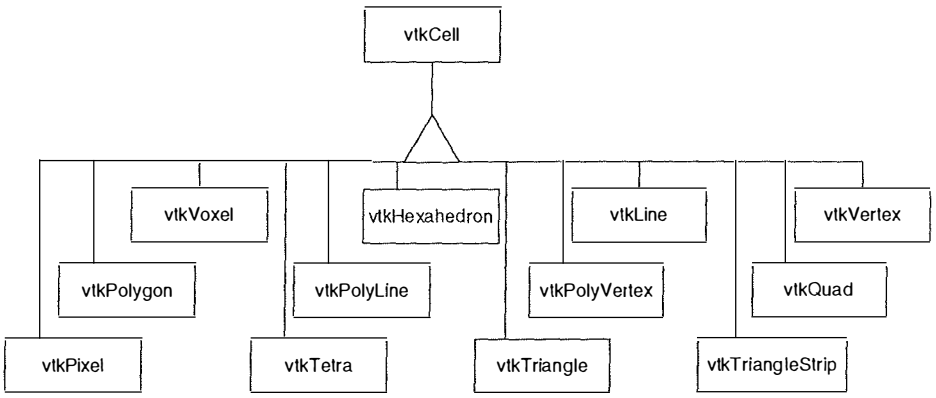
Appendices

Appendix 1 : Object Model for VTK

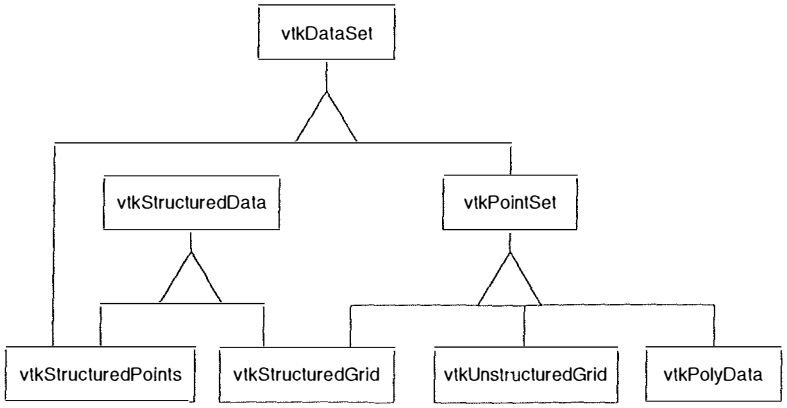
1. The global Object Model



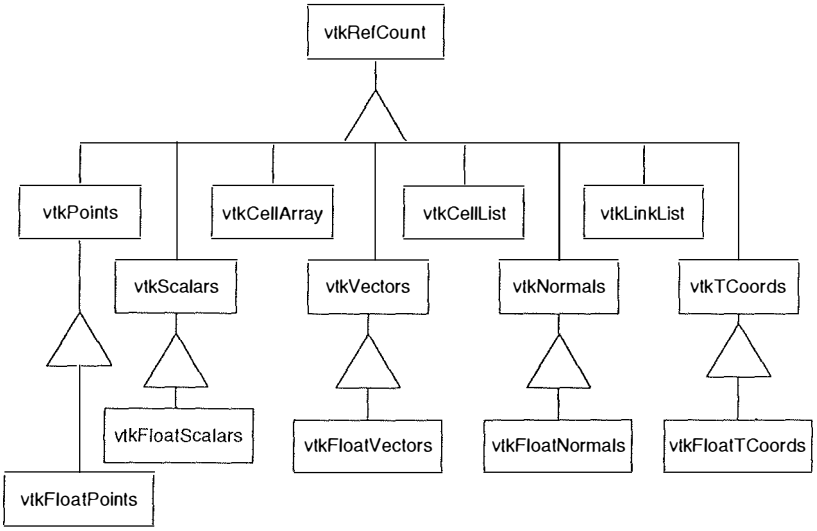
2. The vtkCell Object Model



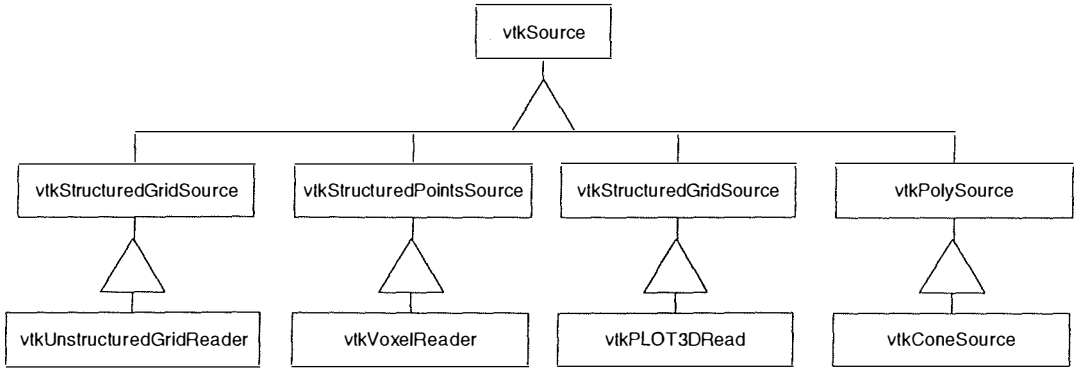
3. The vtkDataSet Object Model



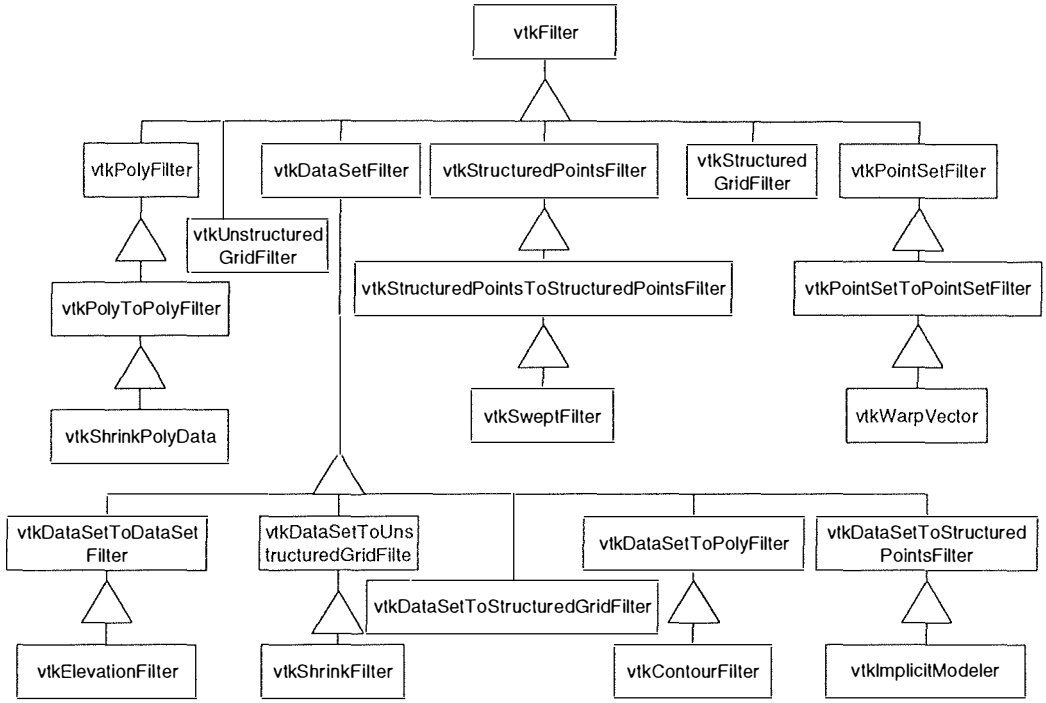
4. The vtkRefCount Object Model



5. The vtkSource Object Model



6. The vtkFilter Object Model


vtkFilter
vtkPolyFilter
vtkDataSetFilter
vtkStructuredPointsFilter
vtkStructured GridFilter
vtkPointSetFilter
vtkUnstructured GridFilter
vtkStructuredPointsToStructuredPointsFilter
vtkPointSetToPointSetFilter
vtkPolyToPolyFilter
vtkSweptFilter
vtkWarpVector
vtkShrinkPolyData
vtkDataSetToDataSet Filter
vtkDataSetToUns tructuredGridFilte
vtkDataSetToPolyFilter
vtkDataSetToStructured PointsFilter
vtkDataSetToStructuredGridFilter
vtkElevationFilter
vtkShrinkFilter
vtkContourFilter
vtkImplicitModeler

Appendix 2 : Quantitative Computed Tomography

The Physical Basis of CT

The ability of CT to detect tiny differences in the x-ray attenuation properties of tissue to be visualized comes from 3 factors:

1. **Signal** : Noise ratio of the data acquisition in CT is less than in conventional radiography due to a greater number of x-ray photons per resolution and the detector utilized exhibiting less noise than radiographic film.
2. **Scattered radiation is reduced** in CT as the beam of radiation is narrowly collimated.
3. **The method of image reconstruction** is unique in CT in that the filtered back projection provides images that are unencumbered by super-imposed underlying and overlying structures that are recorded in conventional tomography.

A narrow beam of x-rays scans across the subject to be imaged in a linear fashion. While traversing the subject the non-absorbed rays are detected by some form of radiation detector that scans synchronously with the beam.

This sequence is repeated at different angles around the subject.

The data acquired consists of a series of "profiles" that reflect the attenuation properties of the subject scanned at different angles.

From these profiles a transverse section of the subject can be constructed.

Generation of the CT Image

The ability of the CT scanner to reproduce the morphology of the assessed structure in the reconstructed image depends on the number of physical measurements taken per unit area.

Analytical determinations have established that the sampling "frequency" in reproducing a structure must be at least 2 to 3 times as fine as the expected resolution of the image. In CT this requires suitable linear and angular sampling by the X-ray beam. To achieve this two configurations are used in the majority of CT scanners:-

- Rotate - Rotate

  The X-ray tube and the array of detectors rotate synchronously about the patient to be imaged.
- Rotate - Stationary Detector Array

  Detectors are arranged in a stationary ring encircling the patient and the X-ray tube rotates around the patient.

The data is then converted by a computer applied algorithm into an "image" that can be displayed as an optical image. CT scanners also incorporate the ability to magnify a portion of the image - zooming.

Such a process does not improve the resolution of the CT image but may render some details more perceptible.

The CT Image

The image generated represents a slice of selected thickness. This is achieved by collimating the beam of X-rays produced; the thickness usually is between 2mm and 1cm depending on the requirements of each clinical study.

The unit most widely used in expressing the attenuation of X-rays in a CT image is known as the "Hounsfield Unit". The unit is defined as:

$$U - \frac{1000Uw}{w}$$

where U is the attenuation coefficient of X-rays of the tissue imaged and Uw is the attenuation coefficient of water. Positive values represent tissues with attenuation values higher than that of water and negative values lower.

As the CT image is recorded in digital form it is relatively easy to apply a number of manipulations to improve the perceptibility of potentially diagnostic information. These include adjustment of the window width and level, magnifying a region of interest or producing sagittal or coronal images.

Appendix 3 : Magnetic Resonance Imaging

Generation and Detection of MRI Signals

Stable nuclei that possess an odd number of protons have the property of a magnetic moment. Therefore if the subject is placed in a strong, uniform magnetic field the effect is that the subject is magnetized very slightly. The magnetic property of the proton has two so-called spin states - one of these positions is lower in energy and more than half are in this stable position. It takes a certain amount of time for polarization to occur. For a simple liquid such as water the process is purely exponential. Tissue water does much the same thing but can have multiple behavior owing to the possibility of varying physiochemical states.

The time constant that is the measure of the rate of the exponential polarization is known as the:

SPIN-LATTICE RELAXATION TIME or T1

The Resonance Condition

The presence of the applied magnetic field forces a precessional motion on the magnetization owing to the spin property of the nuclei. The frequency of the processional motion is the magnetic resonance frequency and is proportional to the strength of the applied magnetic field. This oscillating field can be presented as a voltage if a coil of wire is placed with its axis perpendicular to the field. This oscillating

voltage can then be amplified and the MR signal thereby received. To increase sensitivity the receiver

coil is tuned to be narrowly resonant at the precessional frequency.

To induce an observable signal in the receiver coil a transverse component of the magnetism needs to exist. To accomplish this another coil tuned to the resonance frequency is placed orthogonal to both the axes of the first tuned coil and the magnetic field direction.

The signal does not last forever, it decays exponentially.

This time constant is called the

SPIN-SPIN RELAXATION TIME or T2

For an ideal simple liquid the two time constants T1 and T2 are equal; they are never equal in tissue.

Magnetic Relaxation

The two allowed states for the proton in the presence of the main magnetic field differ in energy, one being more energetically stable than the other - this state being most likeable to all protons.

At the point that the magnetic field is turned off the nuclei are at an elevated energy state. After a transmitter pulse a small amount of energy is absorbed by the nuclei. This energy is re-emitted during relaxation to equilibrium in the form of heat to the surroundings - T1. T2 is where energy is transferred between the protons but does not leave the spin ensemble as a whole. This occurs due to small perturbations in the frequency of procession, with these perturbations being different

for different protons.

As time progresses the protons move less and less in unison and the signal that is induced in the receiver coil becomes weaker.

Signal Processing

The signal induced in the receiver coil is amplified, filtered and the data sent to a computer where a spectrum is obtained by a system of Fourier transformation.

Tissue Differences and Image Contrast

Small differences in tissue properties lead to visible differences in MR signals. A large body of data exists that suggests that the single most consistent contributor to observed differences in relaxation times of protons in tissue is differences with total water content of the tissues.

T1 and T2 both increase with increasing water content - there are other contributing factors in relaxation times such as transition metal ions which have strong magnetic properties.

Each tissue type is likely to have contributions from these and other mechanisms in differing proportions.

MRI allows the formation of a wide variety of transverse, sagittal and coronal images of normal and abnormal anatomy through the use of weak interactions of stable magnetic atomic nuclei.

Appendix 4 : Main classes

```
//################################################
//#                                              #
//# class CScene                                 #
//#                                              #
//################################################
class CScene
{
protected :
     BOOL       AllCuttingPlanesOff,
                AntiAliasing,
                ClockWise,
                CullingFace,
                CuttingPlaneOn[6],
                LightManipulation,
                Translucence,
                SceneBorder,
                ShadeSmooth;
     CCuttingPlane   *CuttingPlaneObject;
     CIntList   *ObjectsHandleList;
     Clight     *Light[8];            // Up to 8 lights
     CObjectList     *Objects;             // List of objects to be drawn
     float      FarPlane,
                MaxObjectSize,
                NearPlane,
                Radius,
                TrX, TrY, TrZ,
                VisionField;          // Champ de vision
     int        BackColor[3],         // Couleur de fond
                LightHandle[8],
                NumberOfLights,
                SceneBorderColor[3],
                WindowHeight,
                WindowWidth;
     HDC        WNDDeviceContext;     // Window Device context
     HGLRC      GLRenderingContext;      // OpenGL Device context
     HWND       ghWnd,                // Parent window
                LastghWnd;            // LastParentWindow

     BOOL       bSetupPixelFormat(HDC hdc);
     void       DefineSceneBorder();
     void       DrawObjectAxis(int Handle);
     void       DrawObjectBox(int Handle);
     float      FindRatio(float SzX, float SzY, float SzZ, float ImportanceRatio);
     HPALETTE   GetOpenGLPalette(HDC hdc);
     void       InitializeLighting();
     int        LoadSurfaceLines(LPCTSTR  ConnectivitiesFileName,  BOOL  ConnectBin,
LPCTSTR CoordonatesFileName, BOOL CoordBin, int Color[3], float DeltaX, float DeltaY,
float DeltaZ, float Importance);
     int        LoadSurfaceTriangles(LPCTSTR ConnectivitiesFileName, BOOL ConnectBin,
LPCTSTR CoordonatesFileName, BOOL CoordBin, int Color[3], float DeltaX, float DeltaY,
float DeltaZ, float Importance);
     int        LoadVolumeLines(LPCTSTR  ConnectivitiesFileName,  BOOL  ConnectBin,
LPCTSTR CoordonatesFileName, BOOL CoordBin, int Color[3], float DeltaX, float DeltaY,
float DeltaZ, float Importance);
```

```
    int       LoadVolumeTriangles(LPCTSTR ConnectivitiesFileName, BOOL ConnectBin,
LPCTSTR CoordonatesFileName, BOOL CoordBin, int Color[3], float DeltaX, float DeltaY,
float DeltaZ, float Importance);
    void            NormalCalculation(float Vertex1[3], float Vertex2[3], float
Vertex3[3], float vout[3]);
    void      SetNumberOfElements(int Handle, int NumElm);
    void      SetNumberOfNodes(int Handle, int NumNod);
    void      VectorNormalization(float Vector[3]);
    void      DrawBis();

public :
    CScene(float  NearPlane,  float  VisionField,  float  MaxObjectSize,  int
BackColor[3]);
    ~CScene();
    void   CreateCuttingPlanes(int Colour[3]);
    int    CreateNewLight();
    void   DeleteContext(HWND  ghWnd);
    void   DrawScene();
    void   GetAmbientAndDiffuseLight(int LightHandle, int Color[3]);
    void   GetBackColor(int BackColor[3]);
    void     GetConnectivityFileInformation(LPCTSTR NodeFile, long &NumberOfNodes,
long &NumberOfElements, int &NumberOfMaterials, BOOL ConnectBin);
    void   GetCuttingPlaneAngles(int PlaneHandle, float &Angle1, float &Angle2);
    void   GetCuttingPlanesParameters(int Color[3], float &Ratio);
    void   GetCuttingPlanesTranslucence(float &Translucence, BOOL &On);
    void   GetCuttingPlanesWires(int &NrOfLines, BOOL &On);
    float  GetCuttingPlaneTranslation(int PlaneHandle);
    void   GetLightPosition(int LightHandle, float &XPos, float &YPos, float &ZPos,
BOOL &Far);
    void    GetMaxObjectDimensions(int Handle, float &Length, float &Height, float
&Depth);
    int    GetNumberOfElements(int Handle);
    int    GetNumberOfLights();
    int    GetNumberOfNodes(int Handle);
    void   GetObjectColor(int Handle, int Color[3]);
    void    GetObjectDeltaLoading(int Handle, float &DeltaX, float &DeltaY, float
&DeltaZ);
    float  GetObjectRatio(int Handle);
    void     GetObjectRotation(int Handle, float &AngleX, float &AngleY, float
&AngleZ);
    void   GetObjectScale(int Handle, float &ScaleX, float &ScaleY, float &ScaleZ);
    void   GetObjectSpecular(int Handle, int Color[3], float &Shininess);
    void    GetObjectTranslation(int Handle, float &DeltaX, float &DeltaY, float
&DeltaZ);
    float  GetObjectTranslucence(int Handle);
    void   GetPerspectiveParameters(float &NearPlane, float &VisionField);
    void   GetRadius(float &Radius);
    void   GetSceneBorderColor(int Color[3]);
    void   GetSpecularLight(int LightHandle, int Color[3]);
    void   GetWindowDimensions(int &Width, int &Height);
    BOOL   IsAntiAliasingOn();
    BOOL   IsCullingFaceOn();
    BOOL   IsCuttingPlanesCreated();
    BOOL   IsCuttingPlaneOn(int PlaneHandle);
    BOOL   IsDrawLightOn(int LightHandle);
    BOOL   IsDrawSceneBorderOn();
    BOOL   IsFrontFaceCWOn();
    BOOL   IsLightOn(int LightHandle);
```

```
    BOOL   IsShadeSmoothOn();
    BOOL   IsShowingCuttingPlaneOn(int PlaneHandle);
    BOOL   IsTranslucenceOn();
    int    LoadObjectLines(LPCTSTR ConnectivitiesFileName, BOOL ConnectBin, LPCTSTR
CoordonatesFileName, BOOL CoordBin, int Color[3], float DeltaX, float DeltaY, float
DeltaZ, float Importance);
    int        LoadObjectPoints(LPCTSTR CoordonatesFileName, BOOL CoordBin, int
Color[3], float DeltaX, float DeltaY, float DeltaZ, float Importance);
    int      LoadObjectTriangles(LPCTSTR ConnectivitiesFileName, BOOL ConnectBin,
LPCTSTR CoordonatesFileName, BOOL CoordBin, int Color[3], float DeltaX, float DeltaY,
float DeltaZ, float Importance);
    int    LoadVtkObjectLines(LPCTSTR VtkFileName, int Color[3], float DeltaX, float
DeltaY, float DeltaZ, float Importance);
    int      LoadVtkObjectPoints(LPCTSTR VtkFileName, int Color[3], float DeltaX,
float DeltaY, float DeltaZ, float Importance);
    int    LoadVtkObjectTriangles(LPCTSTR VtkFileName, int Color[3], float DeltaX,
float DeltaY, float DeltaZ, float Importance);
    void   MakeCurrentContext(HWND ghWnd);
    void   MakeCurrentContext(HWND ghWnd, HDC hdc);
    void   ManipulateLight(BOOL Manipulation);
    void   NewContext(HWND ghWnd);
    void   NewContext(HWND ghWnd, HDC hdc);
    int    ProcessSelection(int xPos, int yPos);
    void   ReleaseCuttingPlanes();
    void   ReleaseLight(int LightHandle);
    void   ReleaseObject(int Handle);
    void   ResetAllCuttingPlanesRotation();
    void   ResetAllObjectsRotation();
    void   ResetCuttingPlaneRotation(int PlaneHandle);
    void   ResetObjectRotation(int Handle);
    void   Resize(HWND  ghWnd);
    void   Resize(int cx, int cy);
    void   RotateAllCuttingPlanes(float AngleX, float AngleY, float AngleZ);
    void   RotateAllObjects(float AngleX, float AngleY, float AngleZ);
    void    RotateCuttingPlane(int PlaneHandle, float AngleX, float AngleY, float
AngleZ);
    void   RotateObject(int Handle, float AngleX, float AngleY, float AngleZ);
    void   SetAllObjectsScale(float ScaleX, float ScaleY, float ScaleZ);
    void   SetAmbientAndDiffuseLight(int LightHandle, int Color[3]);
    void   SetAxisParameters(int Handle, int Color[3], float Length, BOOL Move);
    void   SetBackColor(int BackColor[3]);
    void   SetCuttingPlaneAngles(int PlaneHandle, float Angle1, float Angle2);
    void   SetCuttingPlanesParameters(int Color[3], float Ratio);
    void   SetCuttingPlanesTranslucence(float Translucence);
    void   SetCuttingPlanesWires(int NrOfLines);
    void   SetCuttingPlaneTranslation(int PlaneHandle, float Pourcentage);
    void    SetLightPosition(int LightHandle, float XPos, float YPos, float ZPos,
BOOL Far);
    void   SetObjectBoxParameters(int Handle, int Color[3], float Size);
    void   SetObjectColor(int Handle, int Color[3]);
    void   SetObjectScale(int Handle, float ScaleX, float ScaleY, float ScaleZ);
    void   SetObjectSpecular(int Handle, int Color[3], float Shininess);
    void   SetObjectTranslucence(int Handle, float Transparency);
    void   SetPerspectiveParameters(float NearPlane, float VisionField);
    void   SetSceneBorderColor(int Color[3]);
    void   SetSpecularLight(int LightHandle, int Color[3]);
    void   ShowAllCuttingPlanes(BOOL Show);
    void   ShowCuttingPlane(int PlaneHandle, BOOL Show);
```

```
     void   ShowObjectAxis(int Handle, BOOL Show);
     void   ShowObjectBox(int Handle, BOOL Show);
     void   ShowSceneBorder(BOOL Show);
     void   TranslateAllCuttingPlanes(float DeltaX, float DeltaY, float DeltaZ);
     void   TranslateAllObjects(float DeltaX, float DeltaY, float DeltaZ);
     void   TranslateCuttingPlane(int PlaneHandle, float DeltaX, float DeltaY, float
DeltaZ);
     void   TranslateObject(int Handle, float DeltaX, float DeltaY, float DeltaZ);
     void   TurnAllCuttingPlanesOn(BOOL On);
     void   TurnAntiAliasingOn(BOOL On);
     void   TurnCullingFaceOn(BOOL On);
     void   TurnCuttingPlaneOn(int PlaneHandle, BOOL On);
     void   TurnDrawLightOn(int LightHandle, BOOL On);
     void   TurnFrontFaceCWOn(BOOL On);
     void   TurnLightOn(int LightHandle, BOOL On);
     void   TurnShadeSmoothOn(BOOL On);
     void   TurnTranslucenceOn(BOOL On);
     void   UndoMakeCurrentContext();
};

//################################################
//#                                              #
//# class CCuttingPlane                          #
//#                                              #
//################################################
class CCuttingPlane
{
protected :
     double Equation[6][4];
     float  Angle1[6],
                  Angle2[6],
                  AngleXRotation[6],
                  AngleYRotation[6],
                  AngleZRotation[6],
                  DeltaX[6],
                  DeltaY[6],
                  DeltaZ[6],
                  Max[6],
                  Min[6],
            Color[3],
                  Translucence,
            Ratio;
     int    NumberOfWires;
     BOOL   ShowOn[6],
                  Activated[6],
                  WiresOn;

     void DefineBackPlaneTranslucend();
     void DefineBackPlaneWires();
     void DefineBottomPlaneTranslucend();
     void DefineBottomPlaneWires();
     void DefineEquations(float Min, float Max);
     void DefineFrontPlaneTranslucend();
     void DefineFrontPlaneWires();
     void DefineLeftPlaneTranslucend();
     void DefineLeftPlaneWires();
     void DefineRightPlaneTranslucend();
     void DefineRightPlaneWires();
```

```
      void DefineTopPlaneTranslucend();
      void DefineTopPlaneWires();

public :
      CCuttingPlane(float Min, float Max, int Color[3]);
      CCuttingPlane(int Color[3]);
      void  Activate(int PlaneHandle, BOOL Activate);
      void  Draw(int Handle);
      void  GetAngles(int PlaneHandle, float &Angle1, float &Angle2);
      void  GetColor(float Color[3]);
      void  GetMinMaxParameters(int PlaneHandle, float &Min, float &Max);
      int   GetNumberOfWires();
      float GetRatio();
      void  GetRotation(int PlaneHandle, float &AngleX, float &AngleY, float &AngleZ);
      float GetTranslation(int PlaneHandle);
      void   GetTranslation(int  PlaneHandle,  float  &DeltaX,  float  &DeltaY,  float
&DeltaZ);
      float GetTranslucence();
      BOOL  IsActivated(int PlaneHandle);
      BOOL  IsShowing(int PlaneHandle);
      BOOL  IsTranslucenceOn();
      void  SetAngles(int PlaneHandle, float Angle1, float Angle2);
      void  SetColor(float Color[3]);
      void  SetMinMaxParameters(int PlaneHandle, float Min, float Max);
      void  SetNumberOfWires(int Number);
      void  SetRatio(float Ratio);
      void  SetRotation(int PlaneHandle, float AngleX, float AngleY, float AngleZ);
      void  SetTranslation(int PlaneHandle, float Pourcentage);
      void  SetTranslation(int PlaneHandle, float DeltaX, float DeltaY, float DeltaZ);
      void  SetTranslucence(float Translucence);
      void  ShowCuttingPlane(int PlaneHandle, BOOL Show);
      void  TurnTranslucenceOn(BOOL On);
};


//################################################
//#                                              #
//# class CLight                                 #
//#                                              #
//################################################
class CLight
{
protected :
      float AmbientLight[4],
            DiffuseLight[4],
                  SpecularLight[4],
                  LightPos[4];
      BOOL  LightOn,
                  DrawLight;

public :
      CLight();
      void GetAmbientLight(float AmbientLight[4]);
      void GetAmbientLight(int &CRed, int &CGreen, int &CBlue);
      void GetDiffuseLight(float DiffuseLight[4]);
      void GetDiffuseLight(int &CRed, int &CGreen, int &CBlue);
      void GetLightPos(float LightPos[4]);
      void GetLightPos(float &PosX, float &PosY, float &PosZ, BOOL &Far);
```

```
      void GetSpecularLight(float SpecularLight[4]);
      void GetSpecularLight(int &CRed, int &CGreen, int &CBlue);
      BOOL IsDrawLightOn();
      BOOL IsFar();
      BOOL IsLightOn();
      void SetAmbientLight(int CRed, int CGreen, int CBlue);
      void SetDiffuseLight(int CRed, int CGreen, int CBlue);
      void SetLightPos(float PosX, float PosY, float PosZ, BOOL Far);
      void SetSpecularLight(int CRed, int CGreen, int CBlue);
      void TurnDrawLightOn(BOOL On);
      void TurnLightOn(BOOL On);
};


//################################################
//#                                              #
//# class CReadMeshFile                          #
//#                                              #
//################################################
class CReadMeshFile
{
protected :
      long         ConnectNumberOfElements,
                        ConnectNumberOfNodes,
                   CoordNumberOfNodes;
      int        ConnectNumberOfMaterials;
      BOOL       ConnectNumerotation,
                       CoordNumerotation;
      float        Coordinates[MaxFileSize][3]; // Array of xyz coord
      ifstream   *ElmTextFile;
      FILE       *ElmBinaryFile;

      void    GetConnectivityHeader(char    Buffer[],    BOOL    &Numerotation,    long
&NumberOfNodes, int &NumberOfMaterials, long &NumberOfElements);
      void GetConnectivityLine(char Buffer[], long Nodes[]);
      void  GetConnectivityLine(char  Buffer[],  BOOL  Numerotation,  long  &L,  long
NumberOfNodes, long Node[], int NumberOfMaterials, int Material[]);
      void    GetCoordinateHeader(char    Buffer[],    BOOL    &Numerotation,    long
&NumberOfNodes);
      void GetCoordinateLine(char Buffer[], BOOL Numerotation, long &L, float &X,
float &Y, float &Z);
      void GetCoordinateLine(char Buffer[], float &X1, float &Y1, float &Z1, float
&X2, float &Y2, float &Z2);
      void GetVtkCoordinateHeader(char Buffer[], long &NumberOfNodes);
      void GetVtkConnectivityHeader(char Buffer[], long &NumberOfElements);
      void ReadLine(ifstream f, char Buffer[], BOOL &End);

public :
      CReadMeshFile();
      void    CloseBinaryFileElements();
      void    CloseTextFileElements();
      void    CloseVtkFile();
      void      GetBinaryNodes3(unsigned long &Node1, unsigned long &Node2, unsigned
long &Node3);
      void      GetBinaryNodes4(unsigned long &Node1, unsigned long &Node2, unsigned
long &Node3, unsigned long &Node4);
      void     GetConnectivityFileInformation(LPCTSTR NodeFile, long &NumberOfNodes,
long &NumberOfElements, int &NumberOfMaterials, BOOL ConnectBin);
```

```
     void     GetFileConnectivityHeader(BOOL &Numerotation, long &NumberOfNodes, int
&NumberOfMaterials, long &NumberOfElements);
     void    GetFileCoordinatesHeader(long &Numerotation, long &NumberOfNodes);
     void      GetMaxXYZCoord(float &MaxX, float &MaxY, float &MaxZ);
     void      GetMinXYZCoord(float &MinX, float &MinY, float &MinZ);
     long           GetSizeConnectivity();
     long           GetSizeCoordinates();
     void      GetTextNodes3(unsigned long &Node1, unsigned long &Node2, unsigned
long &Node3);
     void      GetTextNodes4(unsigned long &Node1, unsigned long &Node2, unsigned
long &Node3, unsigned long &Node4);
     void     GetVtkNodes3(unsigned long &Node1, unsigned long &Node2, unsigned long
&Node3);
     void     GetVtkNodes4(unsigned long &Node1, unsigned long &Node2, unsigned long
&Node3, unsigned long &Node4);
     float     GetXCoord(unsigned long Node);
     float     GetYCoord(unsigned long Node);
     float     GetZCoord(unsigned long Node);
     void    ReadBinaryFileCoordinates(LPCTSTR CoordFile);
     void    ReadBinaryFileElements(LPCTSTR NodeFile);
     void    ReadTextFileCoordinates(LPCTSTR CoordFile);
     void    ReadTextFileElements(LPCTSTR NodeFile);
     void    ReadVtkFileCoordinates(LPCTSTR VtkFile);
     void    ReadVtkFileElements();
};

//################################################
//#                                              #
//# class CConverter                             #
//#                                              #
//################################################
class CConverter
{
protected :
     ifstream  *ITextFile;
     ofstream  *OTextFile;
     FILE      *BinaryFile;
     int       Number;

     void    CloseInputTextFile();
     void    CloseOutputTextFile();
     void      GetConnectivityTextFileHeader(char Buffer[], BOOL &Numerotation, int
&NumberOfNodes, int &NumberOfMaterials, int &NumberOfElements);
     void       GetConnectivityTextFileLine(char Buffer[], BOOL Numerotation, int
NumberOfNodes,  unsigned  int  Nodes[],  int  NumberOfMaterials,  unsigned  int
Materials[]);
     void       GetCoordinateTextFileHeader(char Buffer[], BOOL &Numerotation, int
&NumberOfNodes);
     void     GetCoordinateTextFileLine(char Buffer[], BOOL Numerotation, float &X,
float &Y, float &Z);
     void    ReadLine(ifstream f, char Buffer[], BOOL &End);
     void      SetConnectivityTextFileHeader(char Buffer[], BOOL Numerotation, int
NumberOfNodes, int NumberOfMaterials, int NumberOfElements);
     void       SetConnectivityTextFileLine(char Buffer[], BOOL Numerotation, int
NumberOfNodes, unsigned int Nodes[], int NumberOfMaterials, unsigned int Material[]);
     void       SetCoordinateTextFileHeader(char Buffer[], BOOL Numerotation, int
NumberOfNodes);
```

```
     void      SetCoordinateTextFileLine(char Buffer[], BOOL Numerotation, float X,
float Y, float Z);
     void    WriteLine(ofstream f, char Buffer[]);

public :
     CConverter();
     ~CConverter();
     void    CloseBinaryFile();
     void    CloseTextFile();
     BOOL          ConvertConnectivityFileBinaryToText(LPCTSTR   BinaryFile,   LPCTSTR
TextFile);
     BOOL           ConvertConnectivityFileTextToBinary(LPCTSTR   TextFile,   LPCTSTR
BinaryFile);
     BOOL    ConvertCoordinateFileBinaryToText(LPCTSTR BinaryFile, LPCTSTR TextFile);
     BOOL    ConvertCoordinateFileTextToBinary(LPCTSTR TextFile, LPCTSTR BinaryFile);
     void               GetConnectivityBinaryFileHeader(int    &NumberOfNodes,    int
&NumberOfMaterials, int &NumberOfElements);
     void     GetConnectivityBinaryFileLine(int NumberOfNodes, unsigned int Nodes[],
int NumberOfMaterials, unsigned int Materials[]);
     void      GetConnectivityTextFileHeader(BOOL &Numerotation, int &NumberOfNodes,
int &NumberOfMaterials, int &NumberOfElements);
     void        GetConnectivityTextFileLine(BOOL  Numerotation,  int  NumberOfNodes,
unsigned int Nodes[], int NumberOfMaterials, unsigned int Materials[]);
     void    GetCoordinateBinaryFileHeader(int &NumberOfNodes);
     void    GetCoordinateBinaryFileLine(float &X, float &Y, float &Z);
     void    GetCoordinateTextFileHeader(BOOL &Numerotation, int &NumberOfNodes);
     void     GetCoordinateTextFileLine(BOOL Numerotation, float &X, float &Y, float
&Z);
     BOOL    NewBinaryFile(LPCTSTR FileName);
     BOOL    NewTextFile(LPCTSTR FileName);
     BOOL    OpenBinaryFile(LPCTSTR FileName);
     BOOL    OpenTextFile(LPCTSTR FileName);
     void    ResetNumber();
     void               SetConnectivityBinaryFileHeader(int    NumberOfNodes,    int
NumberOfMaterials, int NumberOfElements);
     void     SetConnectivityBinaryFileLine(int NumberOfNodes, unsigned int Nodes[],
int NumberOfMaterials, unsigned int Materials[]);
     void     SetConnectivityTextFileHeader(BOOL Numerotation, int NumberOfNodes, int
NumberOfMaterials, int NumberOfElements);
     void        SetConnectivityTextFileLine(BOOL  Numerotation,  int  NumberOfNodes,
unsigned int Nodes[], int NumberOfMaterials, unsigned int Materials[]);
     void    SetCoordinateBinaryFileHeader(int NumberOfNodes);
     void    SetCoordinateBinaryFileLine(float X, float Y, float Z);
     void    SetCoordinateTextFileHeader(BOOL Numerotation, int NumberOfNodes);
     void    SetCoordinateTextFileLine(BOOL Numerotation, float X, float Y, float Z);
};

//################################################
//#                                              #
//# class VRMLMaker                              #
//#                                              #
//################################################

class VRMLMaker
{
     ofstream  *OTextFile;
     int       Indentation;
```

```
protected :
     void    SetAmbientAndDiffuseLine(float CRed, float CGreen, float CBlue);
     void    SetAmbientAndDiffuseLine(char Buffer[], float CRed, float CGreen, float
CBlue);
     void    SetCommentLine(char Buffer[], char Comment[]);
     void    SetConnectivityFileLine(char Buffer[], int NumberOfNodes, unsigned int
Nodes[]);
     void    SetCoordinateFileLine(char Buffer[], float X, float Y, float Z);
     void    SetCullingLine(BOOL On);
     void    SetCullingLine(char Buffer[], BOOL On);
     void    SetFileLine(char Line[]);
     void    SetFileLine(char Buffer[], char Line[]);
     void    SetRotationLine(float X, float Y, float Z, float Angle);
     void    SetRotationLine(char Buffer[], float X, float Y, float Z, float Angle);
     void    SetScaleLine(float SX, float SY, float SZ);
     void    SetScaleLine(char Buffer[], float SX, float SY, float SZ);
     void    SetSpecularColorLine(char Buffer[], float CRed, float CGreen, float
CBlue);
     void    SetSpecularColorLine(float CRed, float CGreen, float CBlue);
     void    SetShininessLine(float Shininess);
     void    SetShininessLine(char Buffer[], float Shininess);
     void    SetTranslationLine(float TrX, float TrY, float TrZ);
     void    SetTranslationLine(char Buffer[], float TrX, float TrY, float TrZ);
     void    SetTransparencyLine(float Transparency);
     void    SetTransparencyLine(char Buffer[], float Transparency);
     void    SetVertexOrderingLine(BOOL ClockWise);
     void    SetVertexOrderingLine(char Buffer[], BOOL ClockWise);
     void    WriteLine(ofstream f, char Buffer[]);

public :
     VRMLMaker();
     ~VRMLMaker();
     void    BeginCoordinate3Bloc();
     void    BeginIndexedFaceSetBloc();
     void    BeginSeparator();
     void    BeginVRMLFile(char Comment[]);
     void    CloseFile();
     void    DecrementIndentation();
     void    EndCoordinate3Bloc();
     void    EndIndexedFaceSetBloc();
     void    EndSeparator();
     void    EndVRMLFile();
     void    IncrementIndentation();
     BOOL    NewFile(LPCTSTR FileName);
     void    SetCommentLine(char Comment[]);
     void    SetConnectivityFileLine(int NumberOfNodes, unsigned int Nodes[]);
     void    SetCoordinateFileLine(float X, float Y, float Z);
     void    SetCullingBloc(BOOL On);
     void    SetEmptyLine();
     void    SetMaterialBloc(float AmbientAndDiffuse[3], float Specular[3], float
Shininess, float Transparency);
     void    SetMaterialBloc(int AmbientAndDiffuse[3], int Specular[3], float
Shininess, float Transparency);
     void    SetScaleBloc(float SX, float SY, float SZ);
     void    SetShapeHintsBloc(BOOL ClockWise);
     void    SetTranslationBloc(float TrX, float TrY, float TrZ);
     void    SetXRotationBloc(float Angle);
     void    SetYRotationBloc(float Angle);
```

```
     void    SetZRotationBloc(float Angle);
};

//################################################
//#                                              #
//# class CDisplayBinary                         #
//#                                              #
//################################################
class CDisplayBinary : public CDialog
{
protected:
     virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
protected:
     BOOL ReadBinaryFile();
     void AddCap (BOOL Beginning);
     void DisplayArray();
     void DumpCubeToFile();
     void GetExtension (int Slice, char Extension[5]);
     void InitBigArray();
     void InitCube();
     void InvertRectangle(int x, int y);
     void RedrawGreyRectangle();
     void RedrawRedSquare();
     void SaveBinaryFile();
     void SaveInfoFile();
     void SetForSelection();
     void SetInPicture (int &x, int &y);

     // Generated message map functions
     //{{AFX_MSG(CDisplayBinary)
     afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
     virtual void OnOK();
     afx_msg void OnSaveslices();
     afx_msg void OnOtherviews();
     afx_msg void OnValidate();
     afx_msg void OnPaint();
     virtual BOOL OnInitDialog();
     afx_msg void OnAddcap();
     //}}AFX_MSG
     DECLARE_MESSAGE_MAP()

public:
     BOOL m_Inverted;                  // if red square drawn
     BOOL m_IsFirstPaint;         // if the dlg box is to be drawn entirely
     BOOL m_IsRedInverted;
     BOOL m_ShowSlices;
     BOOL      m_AddCap;
     CButton   m_AddCapCtrl;
     CButton   m_Close;
     CButton   m_OtherViews;
     CButton   m_SaveSlices;
     CButton   m_Summary;
     CButton   m_Validate;
     CDC* Dc;
     CDisplayBinary(CWnd* pParent = NULL);   // standard constructor
     char BigArray[512][512];     // the picture drawn
     CPoint m_TopLeft;
     CProgressCtrl   m_Progress;
```

```
        CStatic    m_CtrlVolume;
        CStatic    m_Left;
        CStatic    m_NbSlices;
        CStatic    m_Range;
        CStatic    m_Reading;
        CStatic    m_Rear;
        CStatic    m_Size;
        CStatic    m_Top;
        CString FileName;
        CString m_SliceFileName;
        CString    m_VolumeFraction;
        double m_Xvalue;
        double m_Yvalue;
        double xdim;
        double ydim;
        double zdim;
        enum { IDD = IDD_DISPLAYBINARY };
        TCube m_Cube;                       // The 50x50x50 cube
        unsigned int DisplayIndex;
        unsigned int m_BoxSize;             // Size of the selection box (should be 50)
        unsigned int StartIndex;
        unsigned int StopIndex;
        unsigned int xmax;
        unsigned int xmin;
        unsigned int ymax;
        unsigned int ymin;
        unsigned long m_BoneVoxels;
};


class CGLWorkDoc : public CDocument
{
protected:
        CGLWorkDoc();
        DECLARE_DYNCREATE(CGLWorkDoc)
        int m_NbViews;
        afx_msg void OnSceneAddanobject();
        afx_msg void OnObjectsRenameanobject();
        afx_msg void OnObjectsSelectanobject();
        afx_msg void OnObjectsRemoveobject();
        afx_msg void OnFileMruFile1();
        afx_msg void OnFileSaveAs();
        DECLARE_MESSAGE_MAP()

public:
        BOOL m_ManipulateLights;
        BOOL m_NewDocument;
        CScene * m_Scene;
        CString GetCurrentObjectPath();
        CString m_ElementFile;
        CString m_NodeFile;
        float m_SpecularObjectShininess;
        int GetNbViews();
        int m_CurrentObject;
        int m_GLObject;
        int m_NewObjectNumber;
        int m_ObjectColor[3];
        int m_SpecularObjectColor[3];
```

```
    TObjStruct m_ObjectNames[5];
    UINT m_NbObjects;
    virtual ~CGLWorkDoc();
    virtual BOOL OnNewDocument();
    virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
    virtual void Serialize(CArchive& ar);
    void AddObject();
    void DecrementNbViews();
    void IncrementNbViews();
    void InitObjectNamesArray();
    void InitScenePointer();
    void PackObjectsArray(int KickedObject);
    void RemoveCurrentObject();
    void RenameCurrentObject();
    void SetCurrentObject(int handle);
    void SetCurrentObjectDisplayType (int displayType);
    void SetCurrentObjectHandle(int handle);
    void SetupScene();
};


class CGLWorkView : public CView
{
protected:
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnCBNComboBoxSelChange();
    afx_msg void OnColorsDefaultcolor();
    afx_msg void OnColorsSetcolor();
    afx_msg void OnColorsSpecularproperties();
    afx_msg void OnCuttingplaneOptions();
    afx_msg void OnDestroy();
    afx_msg void OnDisplayAxis();
    afx_msg void OnDisplayChange();
    afx_msg void OnDisplayLights();
    afx_msg void OnHelpDatainformation();
    afx_msg void OnMove(int x, int y);
    afx_msg void OnQuickaxis();
    afx_msg void OnQuickcolor();
    afx_msg void OnQuicklight();
    afx_msg void OnQuickmoveleft();
    afx_msg void OnQuickmoveright();
    afx_msg void OnQuickrestoreangle();
    afx_msg void OnQuickrotatex();
    afx_msg void OnQuickrotatey();
    afx_msg void OnQuickrotatez();
    afx_msg void OnQuickzoomin();
    afx_msg void OnQuickzoomout();
    afx_msg void OnRotateGo();
    afx_msg void OnRotateGoFaster();
    afx_msg void OnRotateGoSlower();
    afx_msg void OnRotateSetangle();
    afx_msg void OnRotateSetscale();
    afx_msg void OnSceneBackgroundcolor();
    afx_msg void OnSceneOptions();
    afx_msg void OnSceneSetangle();
    afx_msg void OnSceneSetscale();
    afx_msg void OnSceneTranslate();
```

```
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnSpeedFaster();
    afx_msg void OnSpeedSetspeed();
    afx_msg void OnSpeedSlower();
    afx_msg void OnTransformTranslate();
    afx_msg void OnUpdateRotateGo(CCmdUI* pCmdUI);
    afx_msg void OnViewAntialiasing();
    BYTE ColorArray[20][3];
    CGLWorkView();
    DECLARE_DYNCREATE(CGLWorkView)
    DECLARE_MESSAGE_MAP()
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnActivateView(BOOL bActivate, CView* pActivateView, CView*
pDeactiveView);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnDraw(CDC* pDC);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
    void InitColorArray();

public:
    BOOL m_ClockWiseSorting;
    BOOL m_Culling;
    BOOL m_FlatShading;
    BOOL          m_AntiAliasing;
    BOOL          m_bRotate;
    BOOL          m_bTransparent;
    BOOL          m_MaintainAspect;//Must or not the aspects change together
    BOOL          m_ManipulateLights;
    BOOL          m_MoveAxis;
    BOOL          m_ShowAxis;
    BOOL          m_ShowBox;
    BYTE          m_CurrentQuickColor;
    CGLWorkDoc*   GetDocument();
    CScene*       m_Scene;
    CString       m_ElementFile;
    CString       m_NodeFile;
    float m_FieldOfView;
    float     m_NearPlane;
    float         m_BoxSize;
    float         m_CurrentXPos;
    float         m_CurrentYPos;
    float         m_CurrentZPos;
    float         m_MaxAxisSize;
    float         m_SpecularObjectShininess;
    float         m_TransparencyLevel;
    int m_BackColor[3];
    int m_ViewNumber;
    int                 m_AngleValueCX;
    int                 m_AngleValueCY;
    int                 m_AngleValueCZ;
    int                 m_AspectX;// Aspect ratio on X axis
    int                 m_AspectY;// Aspect ratio on Y axis
    int                 m_AspectZ;// Aspect ratio on Z axis
    int                 m_AxisColor[3];
    int                 m_AxisLength;
    int                 m_BoxColor[3];
    int                 m_DisplayType;
```

```
        int                     m_GLObject;
        int                     m_objectColor[3];
        int                     m_PresentLocationX;
        int                     m_PresentLocationY;
        int                     m_PresentLocationZ;
        int                     m_RotationSpeed;
        int                     m_SpecularObjectColor[3];
        TCuttingPlane m_Planes[6];
        TStructLight m_Lights[8];
        virtual ~CGLWorkView();
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        void InitComboBar();
        void InitLightsArray();
        void InitPlanesArray();
        void            GetFloatValues (char Buffer[], double &x, double &y);
        void            GetIntValues(char Buffer[], unsigned int &x, unsigned int &y);
        void            PrepareScene();
        void            Rotate(BOOL bRotate);
        void            SetupScene();
        void            Tick();

};
```

Appendix 5 : VRML

This text is exerpt from [VRMLSGI97] web pages.

VRML is a scene description language, which describes 3D environments over the Net. When you access a URL, a "UniqueResource Locator", containing a VRML world, a file is downloaded into your Web browser. VRML Worlds usually end with the file extention .wrl or .wrl.gz as opposed to .html. When your browser sees a file with the .wrl file extension it tells your computer to launch your VRML viewer.

VRML is an acronym for "Virtual Reality Modeling Language." Just as HTML (Hypertext Markup Language) is a file format that defines the layout and content of a 2D page with links to more information, VRML is a file format that defines the layout and content of a 3D world with links to more information. Unlike HTML, however, VRML worlds are spacious and inherently interactive - filled with objects that react to the user and to each other.

VRML allows for information, including links to other pieces of Web content, to be easily represented in an interactive 3D world. VRML is scalable across platforms ranging from PCs to high-end workstations, and soon, the Mac. VRML is also bandwidth efficient. Intricate, interactive 3D worlds can be described in worlds that are similar in size to HTML pages.

Most of the time when VRML files are large it is because of motion capture data, animation, sound, or video, all of which will be reduced as "streaming media" becomes a reality. Straight VRML files are actually very small, especially if special optimization steps are taken.

VRML, it's pronounced vur'mel and it's not just another plug-in. To a growing community, VRML represents the seeds of a new Web. A Web more like the real world -- experiential, interactive, continuous, and, of course, three dimensional. Its applications span the entire spectrum of both the arts and the sciences. One current application of VRML is on JPL's Mars Pathfinder mission.

VRML 2.0 is transforming the Web into a medium that is less like reading a magazine and more like real life. HTML took the Internet and made it accessible to millions of people who are comfortable with 2D graphical user interfaces. VRML is going to take the Internet and the World Wide Web (WWW) to the next level by making it accessible to the billions of people who would rather watch TV than shuffle application windows. Why VRML?

We naturally organize information spatially. Think of receiving a phone call at your desk. During the call you write down the person's phone number on a Post-It note and stick it off to your left. A week later you go to call that person back and you think "where did I put that phone number." In

your mind, you picture the Post-It and look over to see that it is exactly where you left it. That is the spatial map that we all have in our heads to keep track of this database called the world. VRML is the key that will unlock the power of this natural ability to organize the current chaos of the Web.

Put some order on the current 2D chaos. The current metaphor for the Web is starting to break. Most people have a bookmark list that runs off the bottom of the page. Even if we were clever enough to categorize the list, now it runs off the side of our screens... Also, take a peek at your monitor, most of us have multiple application windows open and are constantly trying to shuffle around to get to what you want. These problems are inherent to organizing information on a 2D surface. There are only so many pixels to go around. With 3D if you need more space you simply move forward, or you turn your head. In 3D you get infinite screen real estate for a finite number of pixels on the monitor.

Find what you weren't looking for, but wanted anyway Real estate agents have long babbled "location, location, location." The value of proximity is high in the real world. Locations infer relationships that we use to organize data. Imagine taking a trip to your favorite restaurant. On your way to the restaurant, you pass by a new bookstore. Being a book lover, this is of great value to you and you go inside. You weren't looking for a bookstore, but finding it was a great diversion. If you had teleported directly from your home to the restaurant, you would have never found the bookstore. The value came from your travel and from the location of the bookstore relative to the restaurant.

Researchers and academia have been looking at 3D for years - with the understanding that "it is just better" said Ed McCracken, CEO of Silicon Graphics. There is no reason that the 3D metaphors that we use in real life cannot be translated to the computer to help us get what we want from technology. And there is no reason we should try to constrain ourselves to the accepted 2D interfaces, just because they are already in use. [VRMLSGI97]